The use of microcomputers in DCS AUTODIN tributaries

Anderson, Gordon Ernest

Monterey, California. Naval Postgraduate School

THE USE OF MICROCUMPUTERS IN DCS AUTODIN TRIBUTARIES

Gordon Ernest Anderson

NAVAL POSTGRADUATE SCHOOL
Monterey, California

THESIS

THE USE OF MICROCOMPUTERS IN
DCS AUTODIN TRIBUTARIES

by

Gordon Ernest Anderson

December 1976

Thesis Advisor: V. M. Powers

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE		READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO.	3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) The Use of Microcomputers in DCS AUTODIN Tributaries		5. TYPE OF REPORT & PERIOD COVERED Master's Thesis; December 1976 6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) Gordon Ernest Anderson		8. CONTRACT OR GRANT NUMBER(s)
9. PERFORMING ORGANIZATION NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Naval Postgraduate School Monterey, California 93940		12. REPORT DATE December 1976 13. NUMBER OF PAGES 118
14. MONITORING AGENCY NAME & ADDRESS(if different from Controlling Office) Naval Postgraduate School Monterey, California 93940		15. SECURITY CLASS. (of this report) Unclassified 15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

Approved for public release; distribution unlimited.

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

Present-day Mode I AUTODIN tributaries utilize large-scale computers such as the IBM 360 series, Burroughs 3500 series, and the Univac DCT 9000. The feasibility of using micro-computers (such as the Intel 8080) for such applications was investigated. It was demonstrated that microcomputers can function as Mode I AUTODIN tributaries at speeds greater than

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601 |
(Page 1)

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20. Abstract (Cont'd)

2400 baud. This fact could result in the replacement of expensive leased equipment with subsequent cost savings and expanded use of AUTODIN in tactical and mobile situations. In addition, new methods of describing communication protocols were explored.

The Use of Microcomputers in DCS AUTODIN Tributaries

by

Gordon Ernest Anderson
Captain, United States Marine Corps
B.S., University of Washington, 1968

Submitted in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE IN COMPUTER SCIENCE

from the

NAVAL POSTGRADUATE SCHOOL
December 1976

ABSTRACT

Present-day Mode I AUTODIN tributaries utilize large-scale computers such as the IBM 360 series, Burroughs 3500 series, and the Univac DCT 9000. The feasibility of using microcomputers (such as the Intel 8080) for such applications was investigated. It was demonstrated that microcomputers can function as Mode I AUTODIN tributaries at speeds greater than 2400 baud. This fact could result in the replacement of expensive leased equipment with subsequent cost savings and expanded use of AUTODIN in tactical and mobile situations. In addition, new methods of describing communication protocols were explored.

TABLE OF CONTENTS

I. INTRODUCTION ---------- 12

II. BACKGROUND ---------- 14

A. AUTODIN OVERVIEW ---------- 14

B. FUNCTIONAL DESCRIPTION OF AUTODIN ---------- 17

1. Modes of Operation ---------- 17

2. Synchronous Idle Pattern ---------- 19

3. Line Block Format ---------- 20

4. Control Characters ---------- 26

5. An Analysis of Block-by-Block Operation ---------- 29

C. REASONS FOR INVESTIGATING MICROCOMPUTERS FOR AUTODIN APPLICATIONS ---------- 33

III. MAKING THE AUTODIN PROTOCOL MORE UNDERSTANDABLE ---------- 36

A. DIFFICULTIES IN UNDERSTANDING THE AUTODIN PROTOCOL ---------- 36

B. THE AUTODIN RECEIVE MACHINE ---------- 39

C. THE AUTODIN TRANSMIT MACHINE ---------- 45

IV. SOFTWARE DESIGN AND IMPLEMENTATION ---------- 50

A. DEFINITION OF THE PROBLEM ---------- 51

B. THE HARDWARE ENVIRONMENT ---------- 53

C. CHOOSING A PROGRAMMING LANGUAGE ---------- 58

D. DESIGNING BY LEVELS ---------- 58

E. THE REQUISITE OPERATING SYSTEM ---------- 60

F. IMPLEMENTING THE RECEIVER AND TRANSMITTER PROCESSES ---------- 63

G. TESTING AND DEBUGGING THE PROGRAM ---------- 65

V. FEASIBILITY TESTING ---------- 68

A. RESULTS OF THE PERIPHERAL DEVICE TEST ------ 68

B. RESULTS OF THE TIMING TESTS ---------- 68

VI. CONCLUSIONS AND RECOMMENDATIONS ---------- 71

APPENDIX A: OPERATIONAL SPECIFICATION FOR AUTODIN TEST PROGRAM ---------- 74

APPENDIX B: ACTUAL TEST MESSAGE ---------- 77

APPENDIX C: TIMING TEST CALCULATIONS ---------- 78

THE AUTODIN TEST PROGRAM ---------- 79

BIBLIOGRAPHY ---------- 117

INITIAL DISTRIBUTION LIST ---------- 118

LIST OF TABLES

I. AUTODIN RECEIVER STATE TRANSITION DESCRIPTIONS ---------- 42

II. AUTODIN TRANSMITTER STATE TRANSITION DESCRIPTIONS ---------- 47

LIST OF FIGURES

1. LINE BLOCK STRUCTURE OF AN AUTODIN MESSAGE CONTAINING 277 TEXT CHARACTERS ---------- 22

2. BLOCK-BY-BLOCK OPERATION IN ONE DIRECTION WITH ASC TRANSMITTING AND TRIBUTARY RECEIVING --------- 31

3. THE AUTODIN RECEIVER STATE DIAGRAM ---------- 41

4. THE AUTODIN TRANSMITTER STATE DIAGRAM ---------- 48

TABLE OF ABBREVIATIONS

ASCII	American Standard Code for Information Interchange
ASC	Automatic Switching Center
AUTODIN	Automatic Digital Network
AUTOSEVOCOM	Automatic Secure Voice Communications
AUTOVON	Automatic Voice Network
baud	(in this thesis) bits per second
DCA	Defense Communications Agency
DCS	Defense Communications System
LMF	Language Media Format
UART	Universal Asynchronous Receiver/Transmitter
USART	Universal Synchronous/Asynchronous Receiver/ Transmitter

AUTODIN CONTROL AND FRAMING CHARACTERS:

ACK1	Acknowledge Number 1
ACK2	Acknowledge Number 2
BP	Block Parity
CAN	Cancel
DEL	Delete
EM	End of Medium
ETB	End of Text Block

ETX	End of Text
INV	Suspected Invalid Message
MC	Mode Change
NAK	Negative Acknowledge
REP	Reply
RM	Reject Your Message
SEC	Security
SEL	Selection Channel Characters
SOH	Start of Header
STX	Start of Text
SYN	Synchronous Idle
WBT	Wait Before Transmitting

ACKNOWLEDGEMENTS

I would like to express my gratitude to my thesis advisor Assistant Professor V. Michael Powers for his invaluable assistance and stimulating guidance in the preparation of this thesis. Thanks are due to Lieutenant (jg) Gail Junge and Senior Chief Radioman Frederick Guth of the Naval Telecommunication Center, Monterey, California, for their enthusiastic cooperation and assistance. Also, I must thank Lieutenant Commander Jane Renninger whose pioneering work in reducing the AUTODIN protocol to transition state machines proved indispensable to this thesis.

Finally, I must extend special thanks to my wife, Norma, without whose patience and understanding this thesis could not have been completed.

I. INTRODUCTION

The purpose of this thesis was to investigate the feasibility of using microcomputers (such as the Intel® 8080) as Mode I block-by-block AUTODIN tributaries. Before embarking on the feasibility study, the AUTODIN was studied carefully to ensure that the problem was completely understood. Chapter II examines this background information, giving an overview and a functional description of the AUTODIN. In addition, the reasons for investigating microcomputers as potential AUTODIN tributary stations are explored.

Difficulty was encountered in understanding all ramifications of the AUTODIN protocol. As a consequence, the protocol was described in terms of a receive machine and a transmit machine, which are described in Chapter III. A step-by-step description of the software design of an AUTODIN test program is given in Chapter IV. Careful definition of the problem, understanding the hardware environment, and using the top-down, modular approach are the points emphasized.

® Intel and Intellec are registered trademarks of the Intel Corporation.

Chapter V describes the results of feasibility testing where the correctness of the AUTODIN test program was verified and timing tests demonstrated that the 8080 CPU could function as an AUTODIN tributary at modulation rates exceeding 2400 baud. Finally, Chapter VI summarizes the conclusions and recommendations of this thesis.

II. BACKGROUND

The Defense Communications System (DCS) consists of three major subsystems: the Automatic Voice Network (AUTOVON), the Automatic Secure Voice Communications Network (AUTOSEVOCOM), and the Automatic Digital Network (AUTODIN). The first two subsystems provide nonsecure and secure voice communications, while the third subsystem, AUTODIN, provides a secure record communication capability. This thesis is concerned with the AUTODIN.

A. AUTODIN OVERVIEW

The AUTODIN functions as a single, integrated, worldwide, high-speed, computer-controlled, general purpose communications network which provides record communications service to the Department of Defense (DOD) and other Federal Government Agencies, such as the Department of State. In addition to providing record communications service via various media (such as printed page, magnetic tape, Hollerith cards, etc.), the AUTODIN is also secure and fully automatic. It was designed, engineered, and programmed to provide responsive and continuous operation, minimal loss of service, and no loss of message traffic.

The AUTODIN is a network which consists of 17 Automatic Switching Centers (ASC's) and numerous tributary stations. Of the 17 ASC's, nine are leased and are located in the continental United States and Hawaii. The remaining eight ASC's are government owned and are located in Europe, the Pacific, and Alaska. Each ASC may have up to 300 tributary stations connected to it. This network of ASC's and tributary stations is able to provide responsive communications by use of a system of four message precedence levels: flash, immediate, priority, and routine. By requiring users of the AUTODIN to curtail their use of the higher precedence levels, and by programming the AUTODIN to handle all message traffic on a precedence basis, it is possible for flash precedence level messages to be switched and transmitted around the world in a matter of a few minutes. This capability for rapid communications greatly enhances the effectiveness of the defense establishment of the United States.

A flash level message interrupts all messages of precedence level immediate or less. Precedence level immediate messages are processed before priority or below level messages; however, the lower precedence level messages are not interrupted. Similarly, priority level messages are processed before routine level messages (without interruption of the routine level messages). By proper selection

of precedence levels, users of the AUTODIN can control the speed at which their messages are propagated through the system, with a lower limit of one to three minutes for flash messages and an upper limit of one to two hours for routine messages.

In addition to the variability of speed of transmission provided by the precedence system, there are two other properties of the AUTODIN which greatly enhance its usefulness. First, there is the capability for multiple addressing. The originator of an AUTODIN message may specify that the message go to one, two or hundreds of addressees. This can be accomplished in two ways: by enumerating the addressees, or (if the addressees are grouped together often) by use of collective addresses. The second additional property of the AUTODIN which enhances its usefulness is the ability to use various media for record communications. For example, the originator of a lengthy supply message might transmit the message from magnetic tape and the message could be received as cards on a card punch at the receiving communication center. Conversely, a small communication center without a card capability could transmit logistical data from paper tape and have it punched as cards at the receiving communication center, thus eliminating the need for keypunching the data at the logistical center.

With such properties as variable speed of transmission, selectable media input and output, and multiple addressing, the AUTODIN has provided flexible, responsive, and reliable record communications for over a decade. In order to understand it more fully, it is necessary to examine it on a more technical and detailed level.

B. FUNCTIONAL DESCRIPTION OF AUTODIN

The AUTODIN is a digital network consisting of ASC's and tributary stations with interconnecting communications channels. Both synchronous and asynchronous operation are employed within the AUTODIN; however, asynchronous operation is permitted only on tributary channels, whereas synchronous operation is permitted on both interswitch trunks and tributary channels. For synchronous operation, the AUTODIN will process information at modulation rates of 75, 150, 300, 600, 1200, 2400, and 4800 baud. For asynchronous operation, modulation rates of 75, 150, and 300 baud are permitted. All synchronous AUTODIN communications channels use the American Standard Code for Information Interchange (ASCII). The basic unit for information transfer in AUTODIN is the line block, several of which are shown in Figure 1.

1. Modes of Operation

There are five modes of operation within the AUTODIN. These are Mode I, which is duplex, synchronous operation with

automatic error and channel controls which allow independent and simultaneous two-way operation; Mode II, which is duplex, asynchronous operation allowing simultaneous two-way operation without automatic error and channel controls; Mode III, which is duplex, synchronous operation with automatic error and channel controls (but with one-way information transfer and the return direction used solely for error control and channel coordination responses); Mode IV, which is a unidirectional synchronous operation which can send only or receive only and does not have automatic error control; and Mode V, which permits duplex asynchronous operation and allows simultaneous and independent two-way transmission but which performs only limited channel coordination and display functions.

From the above descriptions, it should be evident that Mode I AUTODIN is the most efficient and hence most desirable type of AUTODIN. All of the asynchronous modes are limited to modulation rates of 300 baud or less. Thus, for medium or high speed data transfer rates, the synchronous modes (Mode I or Mode III) must be used. Mode III contains an inherent disadvantage in that information transfer (or message transmission) is limited to one direction at a time. Thus, only Mode I AUTODIN offers both high-speed operation and simultaneous and independent two-way transmission of

information. This thesis deals solely with Mode I block-by-block operation. All subsequent discussion of AUTODIN assumes Mode I block-by-block operation. The difference between block-by-block and continuous operation will be discussed in Section II.B.5 of this thesis.

2. Synchronous Idle Pattern

In Mode I AUTODIN operation, whenever information is not being transmitted, synchronous idle pattern must be transmitted at the designated modulation rate. Synchronous idle pattern is an even parity character which is equal to the number 96 hexadecimal (or 10010110 binary). Since synchronous idle is transmitted whenever information is not being sent, the receive side of the AUTODIN logic uses synchronous idle pattern to determine whether or not it is in synchronization. At initialization, the Mode I AUTODIN receiver attempts to detect the synchronous idle character (SYN). After the first SYN is detected, the next three characters are checked for the SYN pattern. If the following three characters are SYN, then the receiver considers itself to be in character frame (or synchronized); otherwise, it repeats the above process, repeatedly attempting to achieve character frame. An AUTODIN transmitter may transmit information only if its receiver is in character frame. Likewise,

an AUTODIN receiver may process incoming information only if it is in character frame.

3. Line Block Format

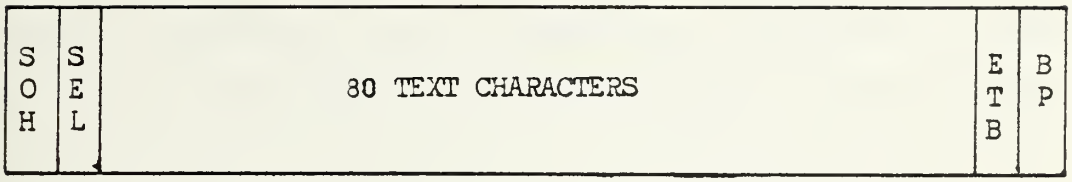
The basic unit for information transfer in AUTODIN is the line block. It may be thought of as a package of information. A typical sequence of events for an ASC transmitting to a tributary station under Mode I operation might be as follows: The ASC sends synchronous idle pattern to the tributary station. The tributary receiver recognizes the synchronous idle pattern and considers itself in character frame. Since Mode I AUTODIN is duplex, the same process takes place (simultaneously and independently) in the opposite direction: the tributary transmitter achieves synchronization with the ASC receiver. Once synchronization has been achieved, it is possible to transmit information in the form of line blocks or "packages" of information. If, for instance, the ASC were transmitting to the tributary, the ASC would send the first line block. If the tributary station received the line block without error, it would reply with an acknowledgement, and the ASC would be free to send a subsequent line block. However, if any error were present in the line block, the tributary would reply with a negative acknowledgement (NAK), and the ASC would retransmit the first line block. In this manner, information is transmitted

in either direction or both directions with channel control and error detection. It should be kept in mind that in Mode I AUTODIN, simultaneous and independent information transfer can occur in both directions. In order to understand more fully the AUTODIN communications protocol, it is necessary to examine the line block structure and associated control characters in detail.

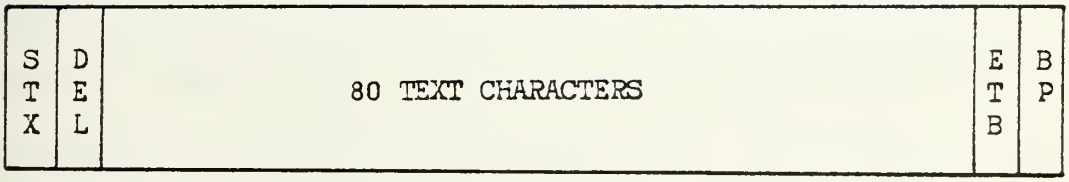
Consider an AUTODIN message which contains 277 text characters or bytes of information. It would be transmitted as four line blocks, the first three of which would contain 80 bytes of information while the fourth would contain 37 bytes of information. Figure 1 shows the line block structure of such a message.

The first character of the first line block of every AUTODIN message is the Start of Heading (SOH) framing character. It is an even parity character which signals the beginning of a new message, and it is always followed by the Select (SEL) framing character. This sequence cannot be split by any other character. The SEL character is an even parity framing character which is always the second framing character of the first line block of every AUTODIN message. Unlike the SOH framing character which is always the same, the SEL character may be one of several alphabetic characters. These alphabetic characters correspond to the

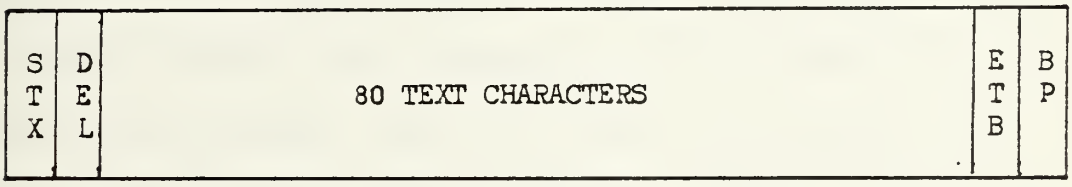
FIRST LINE BLOCK



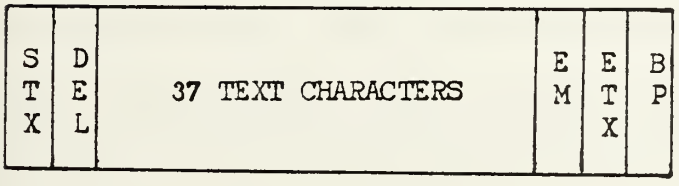
SECOND LINE BLOCK



THIRD LINE BLOCK



FOURTH (LAST) LINE BLOCK



LINE BLOCK STRUCTURE OF AN AUTODIN MESSAGE
CONTAINING 277 TEXT CHARACTERS

FIGURE 1.

various Language Media Format (LMF) indicators but are coded by a different set of characters, according to reference 10. The LMF characters, which appear in the message as it enters and leaves the system, correspond one for one with the SEL characters, which appear in the first line block of the message while it is inside the network. The translation from LMF character to SEL character and back must be accomplished by the network interfaces (tributaries). For example, if an AUTODIN message were narrative in nature, and the originator desired that the addressee of the message receive a printed page version of the message, then the originator would use the LMF indicators "TT." The second "T" would indicate that output on a line printer (or similar device) was desired. This "T" would be translated into the SEL character "H" by the transmitter. Thus, the receiver at an AUTODIN tributary station which received an SOH followed immediately by an even parity "H" would interpret this to mean that the incoming message was to be printed on the line printer. The purpose, therefore, of the SEL character is simply to select the output device at the receiving tributary station. An LMF "C" (meaning card output) would be translated into a "D" SEL character which would cause output on a card punch at the receiving tributary. Reference 2 contains a complete list of SEL and LMF characters.

Following the SOH and SEL framing characters are the first 80 text characters of the AUTODIN message. These characters are transmitted with odd parity. That is, the first seven bits correspond to the American Standard Code for Information Interchange (ASCII) and the eighth bit is either a one or zero such that the total number of ones in the eight-bit byte are odd. The next-to-last character in the first line block of the example in Figure 1 is the End of Transmission Block (ETB) framing character. In fact, ETB is always the third framing character of every line block except the last block of the message. Like all other framing characters, it is an even parity character. The ETB character is immediately followed by the Block Parity (BP) character. No character of any kind may be inserted between ETB and BP. Block Parity is the last framing character of every AUTODIN line block. It may be either odd or even in parity because it is formed by the binary addition without carry (sum modulo 256) of all bytes in the line block. In this way BP serves to check the correctness of received line blocks by detecting single errors.

The second line block of the example message begins with the Start of Text (STX) framing characters. STX is the first framing character of every line block except the first line block which is started with the SOH framing

character. STX is an even parity character which is always followed immediately by a Delete (DEL) framing character, which is also even in parity. The DEL character is the second framing character of every line block except the first one which has an SEL in the second position. The DEL character is used only on links between ASC's and tributary stations. On interswitch trunks between ASC's, the DEL is replaced with a Security (SEC) framing character which is used by the ASC's for the routing of classified and unclassified message traffic.

The remainder of the second line block is the same as the first line block -- 80 text characters followed by the ETB and BP characters. In fact, all subsequent line blocks are the same (STX, DEL, 80 text characters, ETB, and BP) except for the last line block. The last line block begins with STX and DEL framing characters; however, these are followed by 37 text characters and three framing characters. The first of these framing characters is the End of Medium (EM). This even parity character is used to signal the end of an AUTODIN message. It is followed by the End of Text (ETX) framing character (even in parity) and the BP framing character. The BP character is formed as previously described except that it is computed on the 37 text characters and the EM character instead of the 80 text characters as in line blocks one, two, and three.

The line block structure is built and transmitted by the transmit logic of an AUTODIN ASC or tributary. Analogously, the receive logic portion of an AUTODIN ASC or tributary expects to receive information in this line block structure. Now that this structure has been explained, it is possible to discuss the AUTODIN protocol and its associated control characters.

4. Control Characters

In order to provide for channel coordination, control characters are required. Control characters are even parity characters which are always transmitted as contiguous pairs. Six of the most important ones are described below:

(1) Acknowledge Number One (ACK1).

ACK1 is sent by an ASC or tributary to signal the distant transmitter that a line block has been received correctly. ACK1 is the answer to the first line block sent after power-up, or to the first line block received after a message has been cancelled. Thereafter, ACK1 is used alternately with ACK2 to indicate correctly received line blocks.

(2) Acknowledge Number Two (ACK2)

ACK2 is sent as a reply to indicate the correct reception of a line block after a line block has been acknowledged with ACK1. For example, if line block one is received correctly and an ACK1 is sent in reply, then when line block

two is received (correctly), an ACK2 is sent in reply. The sequence of alternate ACK1's and ACK2's is not interrupted between messages; that is, if the answer to last line block of a message was ACK1, then the answer to the first line block of the next message will be ACK2.

(3) Negative Acknowledge (NAK)

Tributaries and ASC's use NAK to signal that a line block has been received with an error in it. NAK is sent after the end of the erroneous line block is received, not at the time the error is detected. Whenever an NAK is received, the transmitting station will retransmit the complete line block to which the NAK applies.

(4) Reject Your Message (RM)

RM's are sent as replies to line blocks. Only an ASC can send an RM, which is sent to the transmitting tributary to signal that there is a defect in the message which cannot be rectified by retransmission of the line block.

(5) Wait Before Transmitting (WBT)

WBT is sent by either an ASC or tributary station in response to a properly framed line block to inform the distant transmitter that the local receiver can no longer accept line blocks. The eventual response to the line block in question may be an ACK1, ACK2, or even NAK; however,

while WBT is being received (and until an ACK or NAK is received), the transmitting station may send only control characters or synchronous idle pattern (SYN).

(6) Reply (REP)

An ASC or tributary station transmitter will use the REP to direct the distant receiver to send its last response or current (updated) response such as ACK1, ACK2, NAK, RM, or WBT. Each transmitter must be equipped with a variable timer hereafter referred to as the answer timer. At the end of each line block transmitted, the answer timer is initialized. When the answer timer expires an REP will be sent if an answer has not been received or if a WBT has been received. Each time an REP is sent the answer timer will be reinitialized. Whenever an ACK1, ACK2, NAK, or RM is received, the answer timer will be stopped. The duration of the answer timer is a function of modulation rate, communication path delays, delays in modems, and receiver response delays. The answer timer duration is determined by adding together all the delays for an expected round trip delay time plus a safety margin. Thus, the answer timer delay is equal to slightly more than the time to receive an expected answer (ACK1, ACK2, NAK, etc.) to a line block or REP. Typical answer timer settings are 3 seconds for 75 to 600 baud circuits, 0.5 seconds for 1200 baud circuits

and 0.25 seconds for 2400 baud circuits. If REP is sent three times in succession without receiving an appropriate reply, an alarm will be sounded.

(7) Cancel (CAN)

CAN is sent by a transmitting station to signal the distant receiver to cancel or discard the current message. The CAN may be initiated manually, automatically by the transmitter upon an incorrectable error condition, or automatically by the receiver whenever an RM is received as the response to a line block.

The aforementioned seven control characters permit channel coordination such that erroneous line blocks are retransmitted, correct line blocks are acknowledged, and, whenever circuit degradation occurs, alarms are activated which bring the requisite human intervention. The next section provides examples which will demonstrate the inter-operative relationship between line blocks, framing characters, and control characters.

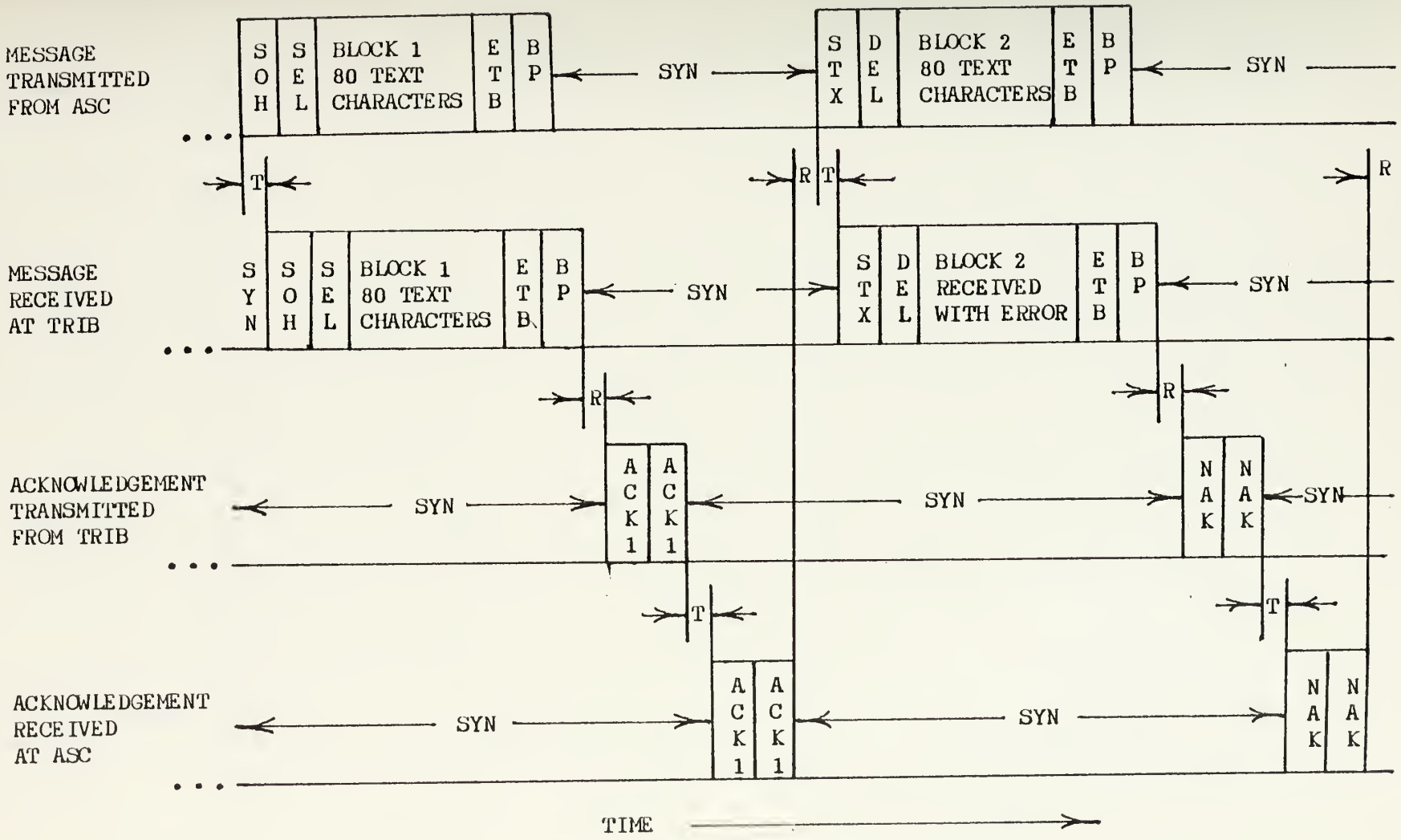
5. An Analysis of Block-By-Block Operation

Within Mode I AUTODIN there are two types of operation: block-by-block and continuous. Under block-by-block operation, a transmitting station sends one line block and does not send a subsequent line block until an ACK1 or ACK2 is received. Under continuous operation, one line block is

sent, then a second one. When continuous operation is working properly, the ACK for the first line block will be received while the second is being transmitted. There is no difference between block-by-block and continuous mode for an AUTODIN receiver and only a trivial change in buffering for an AUTODIN transmitter. This thesis deals only with block-by-block operation.

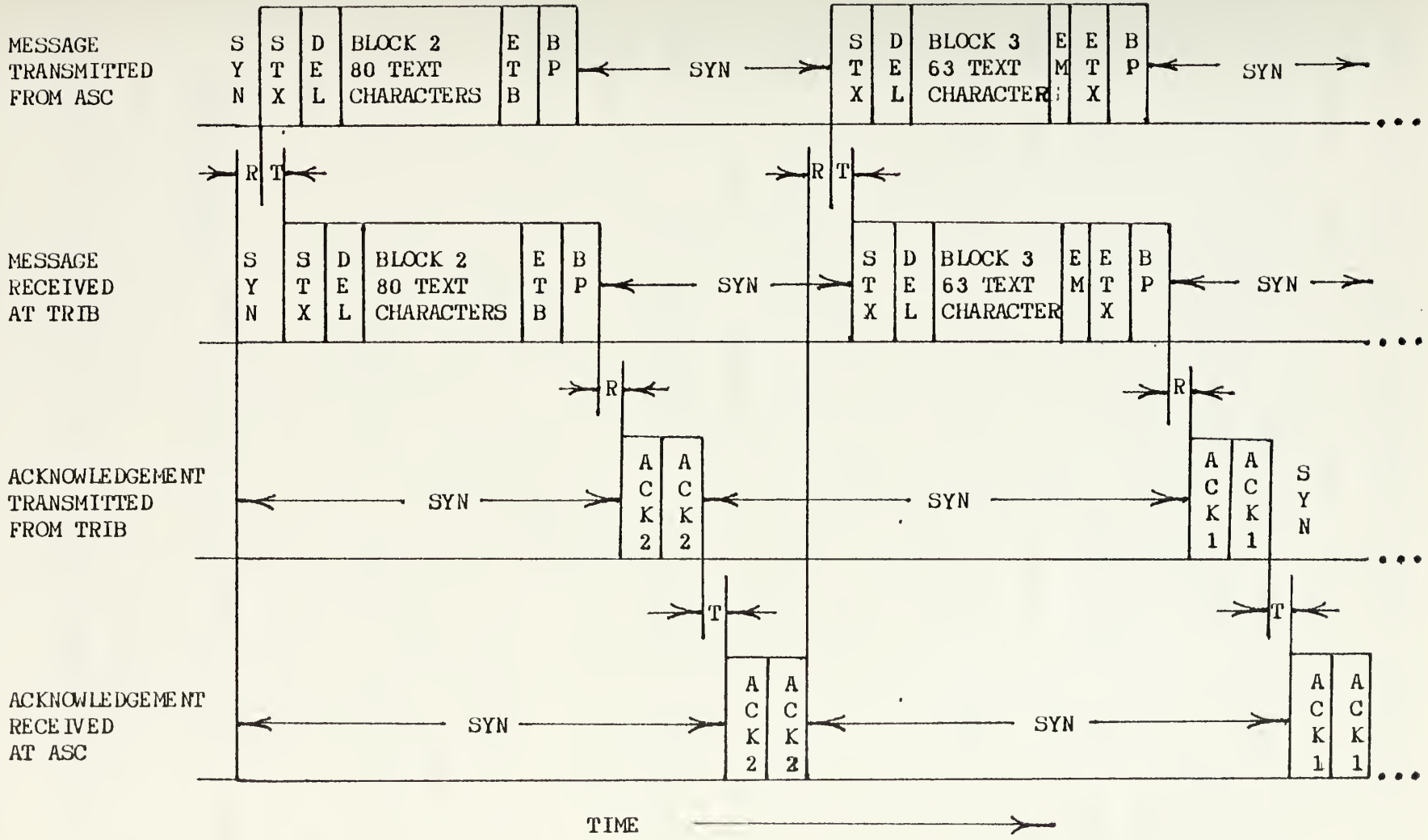
Figure 2 illustrates the AUTODIN protocol: the transmission of data in line block format, the channel coordination obtained from the control characters, the synchronous idle pattern between line blocks and the transmission and response delays involved. The message being transmitted in the example of Figure 2 contains 223 text (or informational) characters. This requires two full-size line blocks of 80 text characters each and a third line block of 63 text characters. In this example, the information transfer is in one direction with the ASC transmitting and the tributary receiving. It will be instructive to trace through Figure 2 from left to right, noting that moving from left to right is analogous to moving forward in time.

Line block one with SOH and SEL for beginning framing characters is transmitted from the ASC and is received at the tributary after a transmission time delay (denoted by "T").



BLOCK-BY-BLOCK OPERATION IN ONE DIRECTION WITH ASC TRANSMITTING AND TRIBUTARY RECEIVING

FIGURE 2a.



BLOCK-BY-BLOCK OPERATION IN ONE DIRECTION WITH ASC TRANSMITTING AND TRIBUTARY RECEIVING
FIGURE 2b.

After the entire line block is correctly received, and after a response delay time (denoted by "R"), the tributary sends two contiguous ACK1's. These are received back at the ASC a transmission time (T) later. The ASC then transmits the second line block of the message; however, this time there is an error in transmission. Consequently, the tributary sends two contiguous NAK's. When these NAK's are received at the ASC, the second line block of the message is retransmitted. This time it is correctly received by the tributary, which sends the appropriate ACK2. Finally, the last line block of the message, which is a short line block containing 63 text characters (marked at the end of text by an EM character), is transmitted. The tributary acknowledges receipt of the last line block with an ACK1, which illustrates the alternation of ACK1's and ACK2's.

From the above descriptions, the reader should have a general idea of how Mode I block-by-block AUTODIN functions. It is merely the transmission, reception, and acknowledgement of line blocks which contain the information to be communicated.

C. REASONS FOR INVESTIGATING MICROCOMPUTERS FOR AUTODIN APPLICATIONS

Reference 1 states that various computers have been approved and certified by the Defense Communications Agency

(DCA) for use as AUTODIN tributary stations. These computers are:

1. IBM 360 Series.
2. RCA SPECTRA 70 Series.
3. Univac DCT 9000 Series.
4. Control Data Corporation CD1700 Series.
5. SOROBAN DST (Mohawk Data Science Corporation).
6. Honeywell 200 Series.
7. ITT World ADX 9300.
8. Burroughs 3500 Series.

Hundreds of the above machines have been installed around the world to provide AUTODIN service to the far-flung units of the Department of Defense. The Naval Telecommunications Center, Monterey, California, is a typical tributary station. It is a 1200 baud Mode I tributary which consists of a Univac DCT 9000 computer with two magnetic tape drives, a card reader, a card punch, a paper tape reader, a line printer, and a communication interface unit. The annual cost to the government to provide this equipment is $67,824.00 per year for equipment leasing and $13,512.00 for on-call maintenance support. Thus, for equipment alone, over $80,000.00 per year must be spent on this tributary station, and this is not an atypical amount. The Communication Center of the Third Force Service Regiment, Fleet Marine Force Pacific, located on the island of Okinawa, costs a similar amount for the same capability: a 1200 baud, Mode I AUTODIN tributary. In this case the equipment is an IBM 360/20 with equivalent peripheral equipment.

In reviewing the above equipment costs, two questions immediately come into mind: First, are such relatively expensive and powerful computers needed for AUTODIN tributary applications? Second, can inexpensive microcomputers function as AUTODIN tributaries? If microcomputers can be programmed to serve as AUTODIN tributary stations, then it is possible to replace the more expensive, powerful machines presently being used and save millions of dollars each year. In addition, since microcomputers are smaller, lighter, and more rugged than the aforementioned large computers, the potential use of microcomputers as AUTODIN tributaries in tactical and mobile situations could greatly improve the record communication capabilities of deployed combat units. In short, greatly reduced costs and expanded AUTODIN service in tactical situations are two potential benefits to be realized if microcomputers are capable of functioning as AUTODIN tributaries. For this reason, the central question of this thesis is: can a microcomputer function as an AUTODIN tributary? If so, how <u>fast</u> can it process information?

III. MAKING THE AUTODIN PROTOCOL MORE UNDERSTANDABLE

Before designing and writing a computer program which would demonstrate the feasibility of using a microcomputer as an AUTODIN tributary, it was necessary to understand completely all of the details of the AUTODIN protocol for Mode I block-by-block operation.

A. DIFFICULTIES IN UNDERSTANDING THE AUTODIN PROTOCOL

Although reference 2 is a very comprehensive and detailed document, it is difficult to use in gaining a complete and precise understanding of the AUTODIN protocol. The major obstacle which prevented an easy understanding of the protocol is the limitation of short-term human memory: it was impossible for the author to digest reference 2 from cover to cover and then suddenly realize and understand the exceedingly complex AUTODIN protocol. The problem was that reference 2 failed to approach the problem of describing AUTODIN from the top down. In other words, instead of giving an overview of AUTODIN and then explaining it in levels of increasing detail, reference 2 appeared to approach the problem from the inside out, a method which was not suitable for rapid and easy understanding of the protocol. This

contention is reinforced by the following example. In 1975, after over a decade of AUTODIN service, the Univac DCT 9000 computer at the Naval Telecommunication Center, Monterey, California, went into the machine halt condition as the result of a software bug which surfaced while an AUTODIN message was being transmitted. Reference 10 specifies that AUTODIN messages shall be terminated by eight line-feeds followed by four N's. However, the DCT 9000, a computer which is sanctioned for AUTODIN use by DCA, interpreted the presence of four contiguous N's in an encoded weather message as the end of message indicator. No line-feeds were involved. This lack of precision in describing the AUTODIN protocol leads to ambiguities which can cause mistakes in programming. The process of describing a complex, detailed protocol in this manner is analogous to describing a building to a blind man brick by brick without first giving a description of the shape, size, and purpose of the building.

For example, when reading reference 2, the author came across the fact that all control characters are transmitted in contiguous pairs. The question then arose as to what the AUTODIN receive logic must do in the event only one control character is received. Should the receiver ignore the character? Should it act upon the character as though it were a valid two-character control sequence? At first, the

author thought that there was an ambiguity on this point; however, the answer was finally found buried in the details of reference 2: the receiver logic ignores single control characters; it only acts upon contiguous pairs of control characters.

This example and others like it served to illustrate the inadequacies of reference 2. What was needed was an overview of the protocol -- some method of describing the inter-operability of all the facets of the protocol. The flowcharts of reference 2 failed to provide an overview of the protocol and also failed to provide enough precision to cover all contingencies. In other words, a better method of describing the AUTODIN protocol was needed. This better method was first used by Renninger in reference 12.

Renninger described the AUTODIN protocol in terms of two state transition diagrams: a receiver and a transmitter. Indeed, throughout reference 2 there are numerous references made to a receiver and a transmitter, but the reader is never told exactly what they are. From studying the work of Renninger, it became clear to the author what the AUTODIN receiver and transmitter were: they were transition state machines which had starting states and which were driven from state to state. Each incoming or received byte represented a potential state transition for the receive machine;

likewise, each byte to be transmitted represented a potential state transition for the transmit machine. The actual transitions made and actions taken depend upon these inputs to the receiver and transmitter and upon the condition of numerous flags which contain detailed information on the overall state of each machine.

The term transition state machine is used here to denote a machine which is derived from and closely related to a finite state machine. The major difference is that while a finite state machine uses only states to define its logic, a transition state machine uses both flags and states. This somewhat more informal method of describing a logical process has two advantages over the finite state machine model. The first of these is that it permits designers to concentrate on the most important states and the second advantage is that the problem can be reduced to an understandable and readable form. The AUTODIN receive and transmit machines are described in more detail in the sections that follow.

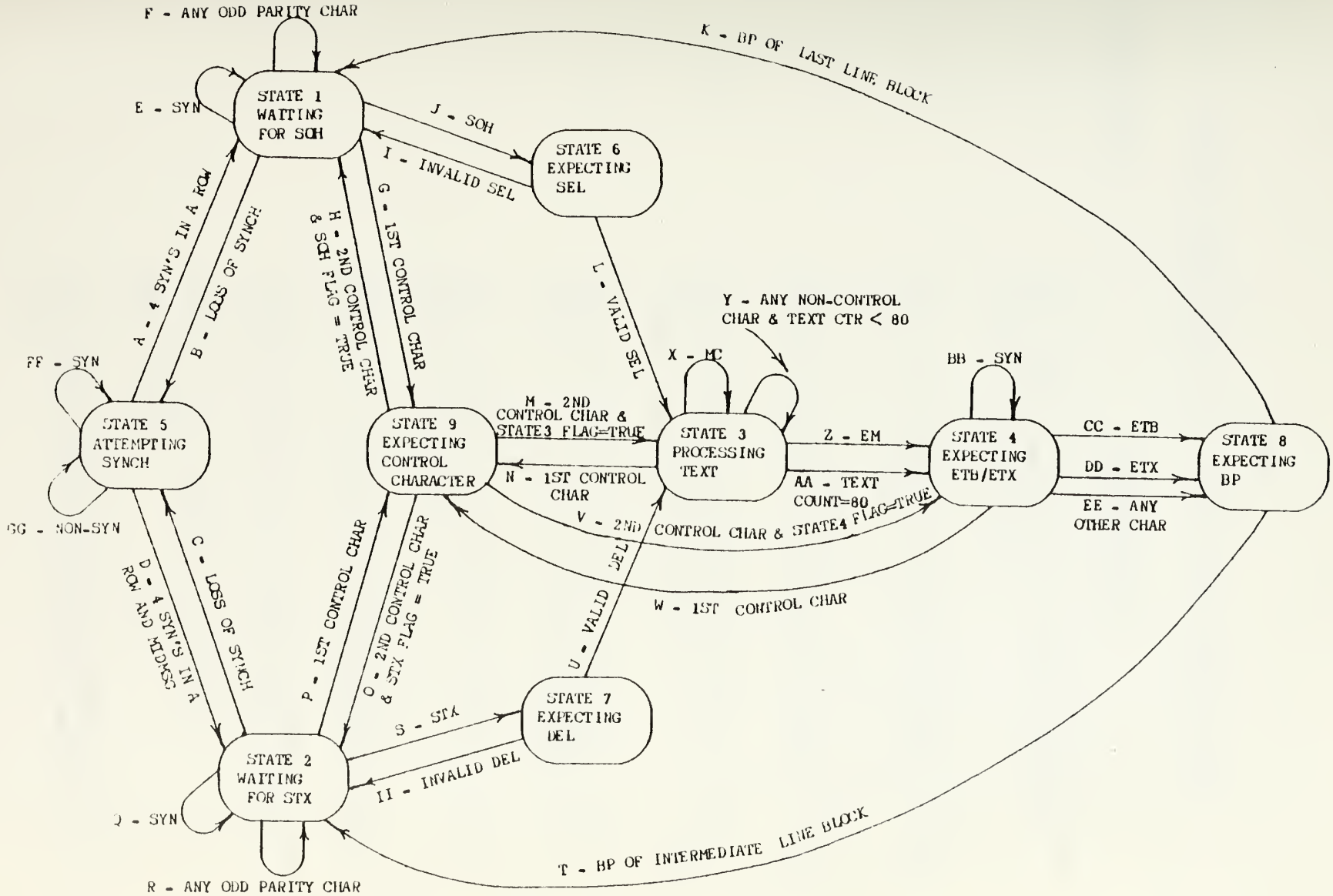
B. THE AUTODIN RECEIVE MACHINE

Renninger described the Mode I AUTODIN receive protocol as a 17-state machine. Here the receive protocol is specified as a nine-state machine. The reason that the protocol can be specified here with eight fewer states is that this

version of the protocol uses more condition flags than Renninger's model, but fewer states. Thus, the two machines are logically equivalent with the following exception: Renninger's machine is for continuous Mode I AUTODIN whereas this machine is for block-by-block Mode I AUTODIN.

Figure 3 depicts the AUTODIN receiver in the form of a nine-state transition diagram. The states are numbered, and the transition paths between states are labeled with letters. A description of each state transition is given in Table I. Each incoming byte or message character corresponds to a transition line on the state diagram, with some transitions beginning and ending in the same state.

It is felt that the state diagram of Figure 3 is a superior method of specifying the control logic of the AUTODIN receiver. It is superior to the flow charts and explanatory text of reference 2 because it utilizes the concept of a finite state machine in its graphical representation to completely specify on one page the receiver protocol. Obviously this is better than scores of pages of text and flowcharts.



THE AUTODIN RECEIVER STATE DIAGRAM
Figure 3

TABLE I

AUTODIN RECEIVER STATE TRANSITION DESCRIPTIONS

Transition	Description
A	Synchronization achieved between messages. Four SYN's received in a row and mid-message flag = false.
B	Loss of synchronization between messages. Receiver timer has expired without 4 SYN's being received and mid-message flag = false.
C	Loss of synchronization between line blocks. Receiver timer has expired without 4 SYN's being received and mid-message flag = true.
D	Synchronization achieved between line blocks. Four SYN's received in a row and mid-message flag = true.
E	SYN character received. Increment syn-counter. If syn-counter = 4 then reset receiver timer and set SYN-COUNTER = 0.
F	Any odd parity character received. Set syn-counter = 0. Check to see if receive timer is expired.
G	First character of a two-character control sequence received. Set SOH flag = true.
H	Second character of a two-character control sequence received and SOH flag = true.
I	Invalid SEL character received.
J	SOH received.
K	BP of last line block in message received (last line block because mid-message = false.

Transition	Description
L	Correct SEL received. Set mid-message flag = true.
M	Second character of a two-character control sequence received and text flag = true.
N	First character of a two-character control sequence received. Set text flag = true.
O	Second character of a two-character control sequence received and STX flag = true.
P	First character of a two-character control sequence received. Set STX flag = true.
Q	SYN character received. Increment syn-counter. If syn-counter = 4 then reset receive timer and set syn-counter = 0.
R	Any odd parity character received. Set syn-counter = 0. Check to see if receive timer is expired.
S	STX received.
T	BP of intermediate line block in message received (intermediate because mid-message flag = true).
U	Correct DEL received. Set mid-message flag = true.
V	Second character of a two-character control sequence and ETB flag = true.
W	First character of a two-character control sequence. Set ETB flag = true.
X	MC received. Set MC flag = true.
Y	Any non-control or non-framing character received and text counter is less than 80. Increment text counter.

Transition	Description
Z	EM received. Set mid-message flag = false.
AA	Text counter = 80.
BB	SYN received.
CC	ETB received.
DD	ETX received.
EE	Any character other than ETX or ETB received. Set error flag = true.
FF	SYN received. Increment syn-counter.
GG	Any character other than SYN received. Set syn-counter = 0.

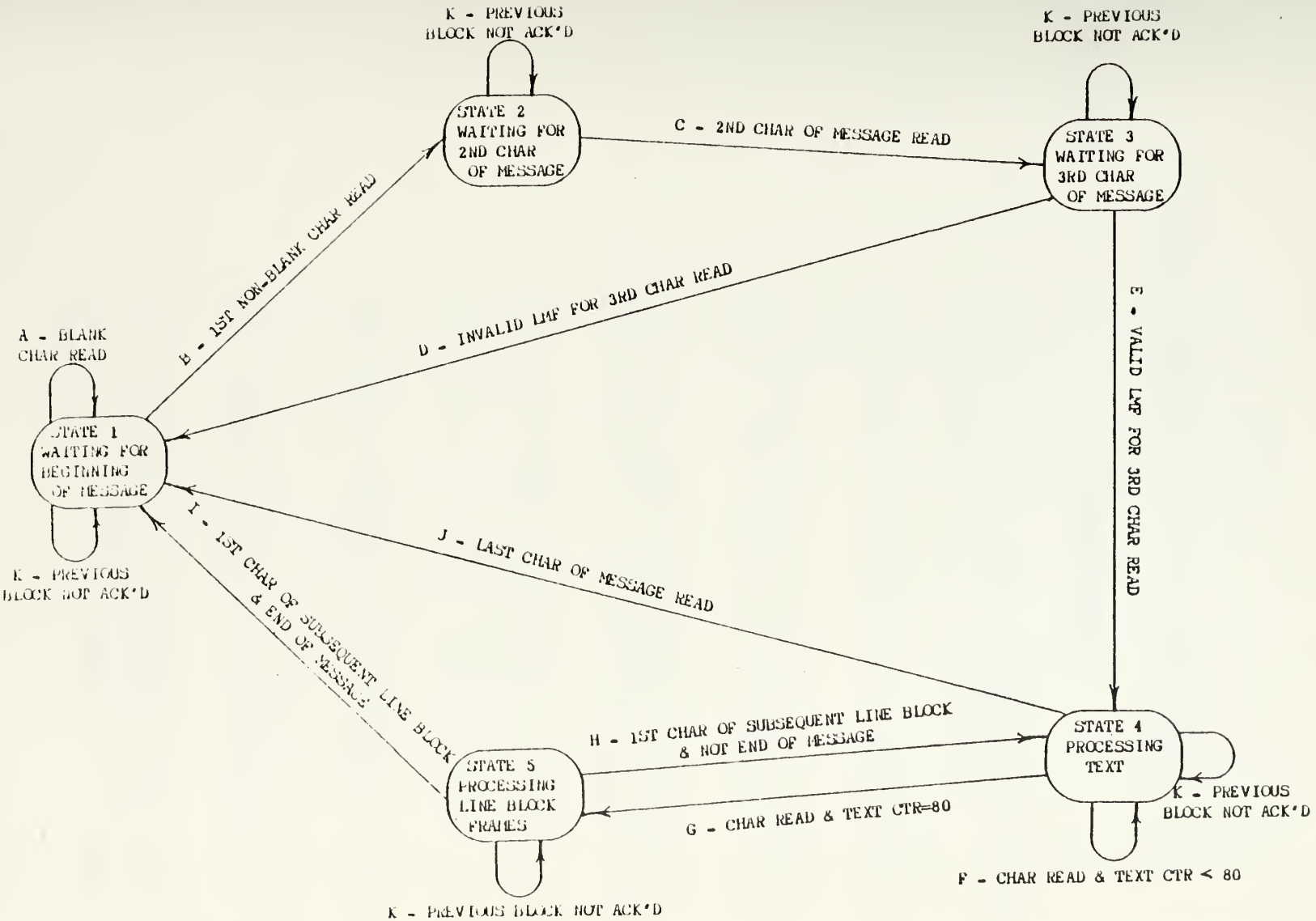
C. THE AUTODIN TRANSMIT MACHINE

The design of the AUTODIN transmit machine is very similar to that of the receive machine except, of course, that the purpose of the transmit machine is to read and transmit characters of text while the purpose of the receive machine is to receive text. Again, Renninger's transmitter consists of nine states whereas the author's machine consists of five states. The reason for fewer states is the same as for the difference in receiver states: fewer states, more condition flags. Finally, this transmit machine and Renninger's differ in that the former is for block-by-block operation and the latter for continuous operation. Otherwise, they are functionally equivalent.

Figure 4 depicts the five-state AUTODIN transmitter as a state transition diagram and Table II gives a description of each of the transitions. Each outgoing byte or text character which is read by the transmit machine corresponds to a line on the state transition diagram. These outgoing bytes can cause transitions from state to state or from a state back to the same state. Again, it is felt that the state diagram method of specifying a communication protocol is far superior to the method used in reference 2.

It should be pointed out that the receive and transmit machines, while specifying the AUTODIN protocol, do not

completely specify all of the details required for implementation on an actual computer. The actual implementation of the receive and transmit machines is the subject of the next section.



THE AUTODIN TRANSMITTER STATE DIAGRAM
Figure 4

TABLE II

AUTODIN TRANSMITTER STATE TRANSITIONS

Transition	Description
A	Blank character read. Blanks are used as leader on paper tape messages and are read and discarded by the transmitter.
B	First non-blank character of a message read. Place character in outgoing line block.
C	Second non-blank character of a message read. Place character in outgoing line block.
D	Third non-blank character of a message read and table lookup indicates invalid LMF character. Cancel the message.
E	Third non-blank character of a message read and table lookup indicates valid LMF character. Place SEL character in outgoing line block and set text counter = 4.
F	Character read and text counter is less than 80. Increment text counter. Place character in outgoing line block.
G	Character read and text counter = 80. Set text counter = 0 and place character in outgoing line block.
H	First character of subsequent line block and end-of-message = false. Set text counter = 2. Place character in outgoing line block.

Transition	Description
I	First character of subsequent line block and end-of-message = true. Place character in outgoing line block.
J	Character read and end-of-message = true. Place character in outgoing line block and set text counter = 0.

IV. SOFTWARE DESIGN AND IMPLEMENTATION

The central issue of this thesis is whether or not a microcomputer can function as a Mode I AUTODIN tributary station and, if it can, at what baud rate can it so function? In order to answer this question it became clear from the very beginning that it would be necessary to develop a computer program which would function as an AUTODIN tributary station. In this way, it would be possible to determine whether or not a microcomputer could perform all of those functions associated with an AUTODIN tributary. If this test proved to be successful, that is, if the microcomputer could perform the necessary tributary functions, then a timing test could be devised which could measure the rate at which the microcomputer could function as an AUTODIN tributary. Consequently, a major portion of the effort expended in this thesis was spent on designing, developing, implementing, testing, and timing a computer program which enabled a microcomputer to function as an AUTODIN tributary station. It should be emphasized, however, that the purpose of the AUTODIN test program is to demonstrate feasibility. It was not designed for actual AUTODIN use.

A. DEFINITION OF THE PROBLEM

The first step in the top-down approach to software development is to define completely the problem to be solved. In this case, the problem was to write a computer program which would enable a microcomputer to function as an AUTODIN tributary station. In addition, the program was to have the property that it could be timed to determine the rate at which it could process AUTODIN messages. This statement defined the problem at its highest level or most general form.

The problem at hand was then taken to the next level of detail. It was determined that the microcomputer, in order to function as an AUTODIN tributary, should be able to interface with the receive side of a communication channel, perform the receive functions of the AUTODIN protocol, and pass the text obtained from the receive channel to a writer device such as a line printer, card punch, or magnetic tape drive. Simultaneously, the microcomputer must also read information from a reader device (such as a paper tape reader, card reader, or magnetic tape drive), put this information into line block format, and interface with the transmit side of a communication channel. In addition, there must be coordination between the transmit and receive functions to provide the full channel coordination and error

detection capability specified and required by the AUTODIN protocol.

The above paragraph represents an important step in the definition of any software problem. That step is specifying the operations which the program must perform. In many circles, this step (or the document which explains it) is called an operational specification. Appendix A is the operational specification for the AUTODIN test program to demonstrate the feasibility of using microcomputers in DCS AUTODIN applications. The reader will note that the operational specification was written in the future tense, since it was developed before the program.

In defining the problem to be solved, two accomplishments served to bring the problem into sharp focus. The first of these was the development of the operational specification of the program. The second was the development of the transmit and receive machine descriptions of the AUTODIN protocol. Indeed, putting the AUTODIN protocol into understandable form was the single most important aspect of defining the problem. The transmit and receive machine descriptions of the AUTODIN protocol appear to be hardware independent; however, many of the points discussed by the operational specification address hardware-dependent problems. For this reason (and in order to achieve an actual

implementation of the AUTODIN protocol) it was necessary to examine the hardware environment in which the program must reside.

B. THE HARDWARE ENVIRONMENT

The Intellec 8/Mod 80 microcomputer development system with its 8080 microprocessor CPU was chosen to develop and test the AUTODIN test program. The first reason for this choice was availability; however, many other reasons also existed. Among these were the wide use of 8080 CPU's (indeed, the 8080 has become an industry standard), the availability of software (such as high-level languages, debuggers, loaders, etc.) for program development and testing, and the ability to address up to 256 peripheral devices. In general, the 8080 is a single-chip, large-scale integrated (LSI) CPU which has 8 and 16-bit registers and can address up to 64K of main memory. References 4 and 5 provide more details on this subject.

The actual microcomputer used for development and testing of the AUTODIN test program was an Intellec 8 microcomputer with 8080 CPU, 16K of main memory, and two input/output (I/O) boards. The first I/O board was configured to permit interfacing with either a teletype or a cathode ray tube (CRT) terminal. The second I/O board was configured to work with

a Universal Asynchronous Receiver/Transmitter (UART). This hardware configuration obviously did not match the normal one found at a Mode I tributary station, which includes magnetic tape drives, card reader, card punch, paper tape reader, and line printer as well as a USART (as opposed to the UART available with the Intellec 8). In addition, in order to assure correctness of the AUTODIN test program, it was decided that tests with actual peripheral devices must be conducted. In order to conduct such tests, an equipment test configuration was developed.

First of all it was decided that the Intellec 8 could be tested back-to-back, with its transmit logic sending to its own receive logic to simulate the full duplex information transfer found on a Mode I communication channel between an ASC and tributary. In fact, as program development progressed, it became obvious that it made no difference whether or not the receiver was receiving information sent by itself (the same computer) or whether it was receiving information from a distant computer. The same was true for the transmitter. Only one minor logical difference became apparent: in using two machines, the receiver, at power-up, would attempt to achieve synchronization before permitting the transmitter to send anything. Obviously, if nothing was ever sent, then SYN characters could never be received. Consequently, for

the back-to-back configuration, a virtual bitstream was programmed into main memory, and part of the initialization of the program would insert SYN characters in this bitstream to achieve initial synchronization. Thereafter, the receive process would fetch bytes from this bitstream just as though it were interfacing with an actual USART. Conversely, the transmit process would insert bytes into this virtual bitstream just as though it were communicating with an actual USART. A side benefit of this method was that it eliminated use of the Intellec 8's UART. The UART was not used for two reasons. First, Mode I AUTODIN calls for synchronous vice asynchronous channel operation. Second, the UART available for testing was configured for seven-bit operation which precluded the use of eight-bit bytes. Eight-bit bytes with odd and even parity are mandatory for the AUTODIN logic. The use of the virtual bitstream concept solved both of these problems and did not cause an adverse effect on the timing considerations since the difference in processing time required to interface with a virtual bitstream and an actual USART is negligible. It is true that with an actual USART, the CPU might have to wait for a byte if the CPU were able to process bytes faster than the USART, or, if the CPU were slower than the USART, then a byte might be missed. However, this contingency was provided for by

conducting worst-case testing (see Section V.B for details). It is interesting to note that an analysis of test results showed that the 8080 CPU must execute an average of 574 instructions per received byte with a virtual bitstream and 573 instructions per received byte for an actual USART. Of these, 572 are identical, demonstrating the negligible difference between the two.

In addition to the virtual bitstream concept, a second aspect of the equipment test configuration had to be carefully thought out prior to programming and testing. This aspect was the matter of peripheral devices. The typical peripheral equipment configuration at a Mode I AUTODIN tributary usually consists of two magnetic tape drives (one for receive, one for transmit), a card reader, a card punch, a paper tape reader, and a line printer. Only two I/O ports, a teletype, and a CRT were available for testing. Since the teletype offered both a print capability for the receive function and a paper tape reader for the transmit function, it was selected over the CRT. The intention was to run tests of the algorithm using the teletype printer and reader simultaneously. This test was needed to check the correctness of the algorithm. However, the program was written so that, on incoming messages, the receiver would examine the SEL character, determine which output

device to select, select the output device, and then write the information on the selected output device (which in this case was always the teletype). In this way, the correctness of the algorithm could be tested without modifying the algorithm which would be used at an actual installation and without modifying the timing considerations. A similar argument holds for the transmit function: the program was made to check I/O ports for ready signals from nonexistent magnetic tape drives and card readers even though any actual input would always take place on the paper tape reader. Incorporating real hardware (such as magnetic tape drives) would require additional device driver routines and additional buffering. These requirements would increase the amount of main memory needed but would have a negligible impact on timing considerations.

By carefully considering all aspects of the hardware configuration prior to writing the program, it was possible to design a program which would be capable of being tested on the existing hardware but which also demonstrated the feasibility of a realistic AUTODIN tributary hardware configuration.

C. CHOOSING A PROGRAMMING LANGUAGE

PL/M, a block-structured, high-level systems language for the 8080 CPU was chosen as the language for developing the AUTODIN test program. There were four major reasons for choosing PL/M. The first reason was that the block structure and other logical constructs (such as if-then-else) facilitated the development of straightforward, efficient algorithms while freeing the programmer from unnecessary details which are often encountered in assembly language programming. The second reason was that, as a systems language, PL/M permits the programmer to control the 8080 just as closely as needed. Third, programs written in high-level languages are much easier to debug and maintain than large assembly language programs. Finally, the use of a high-level language would permit more rapid program development, an important consideration due to time constraints.

D. DESIGNING BY LEVELS

After defining the problem, developing an operational specification, understanding the hardware environment, and choosing a programming language, the next step that was taken was to begin designing the program in levels from the top down to the lowest levels. Much has been written

and spoken about structured programming and the top-down approach; however, in the author's opinion enough cannot be said. The author has used the top-down approach on several medium-size software projects with great success. Applying the approach to the AUTODIN test program also proved to be very successful: the entire project, from conception to successful testing took less than 15 weeks' part-time effort (see Section IV.G for details). It is believed that the reason for this success was due to using the top-down approach and modular, structured programming.

The highest level of the program was designed first, and the most time spent upon it. Correctness was insured at higher levels before proceeding to the design of lower ones. The reader may note that every procedure in the AUTODIN test program was labeled with a design level number. There were five design levels, with level one denoting the highest level and level five the lowest. As the design of the program began at the top level, it was discovered that the receive and transmit machines that were carefully developed in order to understand the AUTODIN protocol did not belong at the highest level of the program but rather at the second level. It became apparent that the actual implementation of these machines would require an operating system at the highest design level of the program to

coordinate and schedule the transmit and receive processes as well as other processes.

E. THE REQUISITE OPERATING SYSTEM

An analysis of the top level program requirements showed that, in addition to the transmit and receive processes, seven other processes were required to implement a functioning AUTODIN tributary. The nine processes are:

1. Receive logic process.
2. Transmit logic process (includes a reader process).
3. Poll peripheral devices process.
4. Poll receive side of USART process.
5. Poll transmit side of USART process.
6. Physical transmit process.
7. Writer process.
8. Operator input process.
9. Operator output process.

The functions of the receive and transmit processes were given in Chapter III of this thesis. The functions of the poll peripheral devices process were to poll the status of the local peripheral devices and to mark the devices as ready or not ready for input or output. Another important process was the poll receive side of USART process whose purpose was to indicate if a newly-received byte were in the USART

and ready for processing. Similarly, the poll transmit side of USART process had as its function to determine if the USART were ready to transmit the next byte. The purpose of the physical transmit process was to actually transfer bytes to the USART for transmission. The purpose of the writer process was to write incoming information onto the selected output device. The operator input process had as its function the input and interpretation of commands from the human operator. Finally, the operator output process had as its function the sending of alarm messages to the operator.

The management of these nine processes was the task of the highest level of the AUTODIN test program. It was necessary for this highest level to schedule the various processes and manage the corresponding peripheral and other devices. This scheduler in algorithmic form is shown below:

```
DO FOREVER:

    CALL POLL USART RECEIVER PROCESS;
    IF RECEIVE LOGIC PROCESS IS SCHEDULED OR
        RECEIVE LOGIC PROCESS DEVICE IS READY
        THEN CALL RECEIVE LOGIC PROCESS;
    IF WRITER PROCESS IS SCHEDULED AND WRITER
        PROCESS DEVICE IS READY
        THEN CALL WRITER PROCESS;
    IF OPERATOR INPUT PROCESS DEVICE IS READY
        THEN CALL OPERATOR INPUT PROCESS;
    IF OPERATOR OUTPUT PROCESS DEVICE IS READY
        AND OPERATOR OUTPUT PROCESS IS SCHEDULED
        THEN CALL OPERATOR OUTPUT PROCESS;
    CALL POLL USART TRANSMIT PROCESS;
```

```
        IF TRANSMIT LOGIC PROCESS DEVICE IS READY
            AND SENDING IS TRUE
            THEN CALL TRANSMIT LOGIC PROCESS;
        CALL POLL PERIPHERAL DEVICES;
    END;
```

The above process scheduler was designed to utilize only polling to determine the status of devices. At first, some consideration was given to handling some of the devices (in particular, the receive side of the USART) on an interrupt basis. This could have been achieved since the 8080 CPU possesses an interrupt capability. However, careful analysis of the problem revealed that no advantage whatsoever was to be obtained from interrupt handling some or all of the devices. The main consideration was speed. When an incoming byte reaches the receive side of the USART, it remains there, ready for plucking by some process, for a time equal to eight times the reciprocal of the baud rate for synchronous operation and ten times the reciprocal of the baud rate for asynchronous operation. If the process scheduler can make one loop (performing all required tasks during this loop) and return to pluck the next byte from the receive side of the USART without ever losing a byte, then it will run fast enough to process a given baud rate. The rate at which the process scheduler can cycle through its DO FOREVER loop will be directly proportional to the baud rate it can handle, and this cycle rate is dependent upon the number of instructions

the CPU must perform per cycle. No gain in speed or efficiency can be obtained by interrupt processing in this case.

The actual implementation of the process scheduler may be found at the end of the AUTODIN program listing labeled program level one. It should be pointed out that the AUTODIN test program runs on the 8080 CPU without a resident operating system. In other words, the program contains its own, built-in operating system functions which consist of the process scheduler at level one of the program and the level five procedures which handle the actual input and output of the bytes. Program levels two, three, and four represent the various logic levels of the AUTODIN protocol and its associated processes such as writer, operator input, etc.

F. IMPLEMENTING THE RECEIVER AND TRANSMITTER PROCESSES

The next task to be performed in developing the AUTODIN test program was to implement the receiver and transmitter processes. These processes were well defined in Chapter III. Consequently, the task of implementing them was greatly simplified.

The receiver process (or RECEIVE$LOGIC, as it was called in the AUTODIN program) was designed to be a nine-state machine and was implemented as a level two procedure which

consisted of a nine-part case statement. Each invocation of the procedure corresponds to waking up of the receive logic process. Based on the input of a newly-received byte and the condition of various flags, the receive logic will perform designated actions and will make a state transition before going back to sleep.

The transmit process (or TRANSMIT$LOGIC, as it is called in the AUTODIN program) was designed to be a five-state machine and was implemented as a level two procedure which consisted of a five-part case statement. Each state (or case) was implemented as a level three procedure.

It is instructive to compare the state transition diagram of Figure 4 with the actual program as given in the listing. The procedure XMT$STATE$3 (contained in procedure TRANSMIT$LOGIC) corresponds to state three of Figure 4. One of two possible transitions will be made from state three. If the byte just read from the selected input device corresponds to a correct LMF character, then the transmit logic will place that character in the third text slot of the outgoing line block, perform a table lookup to find the corresponding SEL character, and place the SEL character in the second framing position of the outgoing line block. Then, the transmit logic will set its new state to four and go to sleep until reawakened by the process scheduler. On

the other hand, if the newly-read byte does not match with a correct LMF character then the transmit logic will send an alarm to the operator, cancel the current message, set its new state to one (the start state), and go to sleep.

The fact that the program was designed in levels is illustrated by pointing out that in this example the job scheduler and device manager are at level one, the transmit logic process is at level two, the actions of transmit state three are at level three, the procedure which checks LMF's for transmit state three is a level four, and the simple procedures which actually input and output bytes are at level five.

G. TESTING AND DEBUGGING THE PROGRAM

The testing and debugging of the AUTODIN test program was performed with relative ease, a fact the author attributes to the top-down approach. Of the 15 weeks spent on the project (from inception to successful testing), seven were spent defining the problem and designing the uppermost levels of the program, five were spent in coding and program development, and three were spent in testing and debugging the program on the Intellec 8. The definition of the problem and design of the upper levels have been discussed previously and consequently will not be discussed here.

Coding and program development were greatly facilitated by the use of an interactive, time-share terminal connected to the IBM 360 system of the Naval Postgraduate School. This terminal provided three invaluable tools for program development: a powerful context editor, a PL/M compiler, and an 8080 simulator (called Interp 80). These tools facilitated rapid program development and permitted the design-by-level approach by allowing testing of program modules at each level of development. Interp 80 was particularly useful in program development. For example, the AUTODIN receiver logic was tested to see if it could correctly recover from error conditions (such as incorrectly received line blocks). Using Interp 80, it was simple to introduce errors in order to test the performance of the receiver logic under various error conditions. Of the five bugs found during program development, four were found using Interp 80 before attempting to test on the Intellec 8. The five program errors discovered were contained in a program of over 1700 lines of source code. This translates into approximately one error per 350 lines of code -- proof that the top-down approach can produce good software.

In addition to the above problem, a timing problem was encountered during testing with the teletype. This was caused by the extremely slow reaction speed of the teletype

as compared to the 8080 CPU. The problem was rectified by inserting delays into the program. Upon completion, the object program was approximately 6100 bytes in size.

V. FEASIBILITY TESTING

As previously mentioned, the AUTODIN test program was designed to be tested in two ways. The first test was performed with actual peripheral devices to test the correctness of the algorithm, and the second test was performed with all devices virtual to obtain timing results on the 8080 CPU. Changing from one type of testing to the other was accomplished by changing one line of source code.

A. RESULTS OF THE PERIPHERAL DEVICE TEST

This program demonstrated its ability to simultaneously input and output information using the teletype printer and paper tape reader. In addition, the program demonstrates its ability to send alarm messages to the operator. Appendix B shows an actual test message which was sent on the Intellec 8.

B. RESULTS OF THE TIMING TESTS

Three timing tests were conducted. In each of them, 180,000 bytes were processed, and the time required for this to be done was recorded. This was actually accomplished by starting the program, using a stopwatch, and having the 8080 go into machine halt after 180,000 received bytes.

Appendix C shows the calculations used to obtain baud rates from these measurements.

The first timing test consisted of running the program with virtual peripheral devices for 180,000 bytes to obtain an average baud rate. During the test, message traffic was always being transmitted and received. Thus, during each cycle of level one, the program was required to receive one byte, write one byte, read one byte, and transmit one byte (in addition to polling all peripheral devices, even though they were not used). The result of this timing test was 3354 baud.

The second timing test was exactly the same as the first one with one difference. In order to make the AUTODIN test program capable of being run with both actual and virtual peripheral devices, it was necessary to make numerous checks throughout the program for the virtual or actual conditions. This required additional time. Consequently, these checks were removed, and the program was again timed. This time, the result was 3723 baud, a slightly faster rate, as expected.

The third timing test took into account worst case conditions as opposed to the average conditions of the first two tests. This test was necessary because the virtual USART was used. When using a virtual USART, no received byte is ever lost. Thus, even though an average baud rate of 3723

was measured, there might be worst case conditions where the receiver logic was going through its worst case (most time-consuming) processing coincident with the transmit logic doing the same, while at the same time, the operator input, operator output, and writer processes all required attention. Analysis of the AUTODIN test program revealed that, for the receiver logic, performing state nine actions (second control character of a two-character control sequence) were most time-consuming. For the transmit logic, state three actions (performing a LMF lookup and LMF-to-SEL conversion) were the most time-consuming. These actions were more time-consuming than error recovery. Under these conditions, an actual USART running at 3723 baud would result in lost received bytes. Therefore, it was necessary to conduct a worst-case test of the AUTODIN test program where these most time-consuming actions were repeatly performed. The result of this test was 2785 baud. An additional result was that, using an elapsed time of 517 seconds (See Appendix C) and an average instruction time of five microseconds, it was determined that the 8080 CPU executed an average of 574 instructions per received byte.

VI. CONCLUSIONS AND RECOMMENDATIONS

The AUTODIN test program is not an item of software ready for installation in AUTODIN tributary stations around the world. Rather, it was designed to demonstrate the feasibility of using a microcomputer to perform all of the functions required of a Mode I AUTODIN tributary station. In this regard, the AUTODIN test program fulfilled the purpose for which it was designed. By using conservative analysis techniques and taking into account worst-case processing requirements, it was shown that the 8080 CPU can perform all of the functions associated with an AUTODIN tributary station at modulation rates of 2400 baud. Furthermore, the AUTODIN test program required approximately 6000 (8-bit) words of main memory. Part of this memory requirement came from test parameters which need not be present in an actual working program. On the other hand, larger buffer sizes for interfacing with actual magnetic tape drives might be desirable. In addition, more main memory for certain nice-to-have features such as strings containing classification headings would be required. Nevertheless, it is conservatively estimated that 8194 words of main memory would handle the requirements for a fielded, working version of the program.

The result of all this is that AUTODIN communications can join the microcomputer revolution, and the revolution can be joined at a respectable baud rate of 2400. An 8080 CPU costs less than 30 dollars. An 8080 CPU, 8194 words of memory, power supply, cabinet, and I/O boards (with USART) cost less than one thousand dollars. There is absolutely no reason for continuing to lease expensive, large-scale computers at 60-80 thousand dollars per annum. It is true that the cost of peripherals must be added to the low cost of an 8080 based microcomputer system, but even with these costs added, the potential cost savings to the Federal Government are phenomenal. It is recommended that immediate attention be given to the official sanctioning and qualifying of microcomputers as DCS-approved equipment for AUTODIN use.

Another important implication of this thesis is the potential use of AUTODIN tributaries in mobile and tactical applications. Since microcomputers are so small and lightweight, they can be mounted in vehicles and aircraft to provide access to a worldwide digital information network. As defense management and weapon systems become more and more complex, the requirements for information in all forms (printed page, magnetic tape, floppy disk) at lower and lower echelons of command will increase. The use of microcomputers

will make it possible to expand the number of tributaries, giving more commanders at lower levels rapid access to the DCS. The field teletype can be replaced with a microcomputer connected to a lightweight line printer and (perhaps) a floppy disk unit, which will greatly improve the throughput rate and flexibility of communicated information. These are only a few of the potential applications. It is recommended that future development of field record communication systems take into account the use of microcomputers.

Finally, a side-product of this thesis was the state transition method of describing the AUTODIN protocol. This method proved to be vastly superior to the method used by DCA to describe AUTODIN. It is recommended that the state transition model be researched further, for it is felt that, with refinement, it could become a most effective method of describing communication protocols.

APPENDIX A

OPERATIONAL SPECIFICATION FOR AUTODIN TEST PROGRAM

I. SYSTEM OVERVIEW

The purpose of the test program shall be to investigate the feasibility of using a microcomputer such as the Intel 8080 (or equivalent) as a Mode I block-by-block AUTODIN tributary station. To demonstrate feasibility, it will be necessary to program the microcomputer to perform all of the functions that a tributary normally performs. These functions include the duplex, simultaneous transmission and reception of information via input from magnetic tape, card, or paper tape and output via line printer (or teletype), card, or magnetic tape. In the feasibility demonstration, the aforementioned peripheral devices may be real or virtual. In addition, the simultaneous transmission and reception of information over a full-duplex communication channel via a Universal Synchronous/Asynchronous Receiver/Transmitter (USART) must be accomplished. Furthermore, appropriate messages to the human operator must be sent whenever necessary. Although simultaneous transmission and reception is required, only one input device and one output device (which may be of the same or different type) may be selected

and in use at any point in time. In fact, the input device remains the same for each AUTODIN message transmitted; likewise for received messages and output devices. For example, the system might be simultaneously transmitting information from cards and receiving information which was being printed on a line printer.

In addition to performing the above tasks, the program must be designed such that the correctness of the algorithm may be tested by interfacing with actual peripheral devices. On the other hand, the program must be capable of being easily changed to work with virtual peripheral devices so that timing tests may be conducted. The reason for using virtual peripheral devices for timing tests is so that the central processing unit (CPU) of the microcomputer may run at full speed: the purpose of the program is to determine the speed at which a microcomputer can process AUTODIN messages and not to determine which input/output devices are rapid enough to function at AUTODIN tributaries.

III. PROCESSING REQUIREMENTS

A. RECEIVE PROCESSING

Incoming information arrives at the USART via a communication link and is transferred to the microprocessor, examined for parity correctness, stripped of control and

framing bytes, and transferred to the output device selected according to the SEL character in the incoming message. Acknowledgements (ACK1/ACK2) will be sent for correctly received line blocks; negative acknowledgements will be sent for incorrectly received line blocks. Synchronous idle will be recognizable by the receiver function, and notification of any loss of synchronization will be displayed to the operator.

B. TRANSMIT PROCESSING

When the human operator activates an input peripheral device for transmission by mounting a paper or magnetic tape or by loading a card deck into a card reader, the program must recognize that transmission is to begin. The program must begin transmission by reading the selected input device, building the line blocks for transmission, and must actually transmit the information, byte by byte, to the USART. Included in this operation is the insertion of proper parity and framing characters. In addition, the requisite coordination between the transmit and receive functions must occur so that proper channel coordination takes place according to the AUTODIN protocol.

APPENDIX B

ACTUAL TEST MESSAGE

```
RTTUZYUW RUWJAGC0000 2122200-UUUU--RUWJAGC.
ZNR UUUUU
R 302200 JUL 76
FM NTCC MONTEREY CA
TO NTCC MONTEREY CA
BT
UNCLAS//N00000//
THIS TEST MESSAGE DEMONSTRATES THE ABILITY OF THE 8080
CPU AND INTELLEC 8 MICROCOMPUTER TO PROCESS AUTODIN
MESSAGES.  ALTHOUGH THIS PARTICULAR MESSAGE IS PRINTED
ON A TELETYPE, THE 8080 IS CAPABLD OF INTERFACING WITH
OTHER PERIPHERAL DEVICES SUCH AS LINE PRINTERS, MAGNETIC
TAPE DRIVES, AND CARD PUNCHES/READERS.  THE POTENTIAL
USE OF MICROCOMPUTERS FOR AUTODIN APPLICATIONS HAS TWO
MAJOR IMPACTS:

     1)  CONSIDERABLE COST SAVINGS MAY RESULT FROM
         USING MICROCOMPUTERS IN PLACE OF LARGER,
         MORE EXPENSIVE COMPUTERS.

     2)  THE POSSIBLE USE OF LIGHTWEIGHT, RUGGEDIZED
         COMMUNICATIONS TRIBUTARIES FOR USE IN
         TACTICAL SITUATIONS CAN GREATLY IMPROVE
         FIELD RECORD COMMUNICATIONS.

END OF TEST MESSAGE.
BT
#0000



NNNN
```

APPENDIX C

TIMING TEST CALCULATIONS

TEST 1: CONSTANT CHECKING FOR VIRTUAL/ACTUAL DEVICES

ELAPSED TIME = 429.4 seconds

$$\frac{180{,}000 \text{ bytes}}{429.4 \text{ seconds}} \times 8 \text{ bits/byte} = 3354(\pm 25) \text{ baud}$$

TEST 2: NO CHECKING FOR VIRTUAL/ACTUAL DEVICES

ELAPSED TIME = 386.8 seconds

$$\frac{180{,}000 \text{ bytes}}{386.8 \text{ seconds}} \times 8 \text{ bits/byte} = 3723(\pm 25) \text{ baud}$$

TEST 3: WORST CASE PROCESSING

ELAPSED TIME = 517.0 seconds

$$\frac{180{,}000 \text{ bytes}}{517.0 \text{ seconds}} \times 8 \text{ bits/byte} = 2785(\pm 25) \text{ baud}$$

```
100H:    /*  THE PROGRAM WILL BE LOADED IN THE 8080
             BEGINNING AT 100 HEX     */

    /*                    A U T O D I N

PROGRAMMER: GORDON E. ANDERSON.
SOURCE COMPUTER: IBM 360 (USED FOR COMPILATION).
OBJECT COMPUTER: INTELLEC 8 (USED FOR EXECUTION).
CLASSIFICATION: UNCLASSIFIED.
SOURCE LANGUAGE: PLM.
DATES: AUGUST - NOVEMBER 1976.

    THE PURPOSE OF THIS PROGRAM IS TO DEMONSTRATE THE FEASABILITY
OF USING A MICROCOMPUTER, SUCH AS THE INTELLEC 8 WITH 8080 CPU, FOR
A DEFENSE COMMUNICATION SYSTEM (DCS) MODE I AUTODIN TRIBUTARY STATION
(BLOCK BY BLOCK MODE OF OPERATION).  THE PROGRAM IS DESIGNED TO PERFORM
ALL OPERATOR INTERFACES, LOCAL PERIPHERAL INTERFACES, COMMUNICATION
INTERFACES, AS WELL AS THE LOGICAL PROTOCOL INTERFACE WITH AN
AUTOMATIC SWITCHING CENTER (ASC).  THE PROGRAM WILL PERFORM THIS
INTERFACING INDEPENDENT OF THE RATE OF INFORMATION TRANSFER
(BAUD RATE OR BITS PER SECOND).  THIS WILL PERMIT TESTING
OF THE PROGRAM AT DIFFERENT BAUD RATES IN ORDER TO DETERMINE
THE MAXIMUM SPEED AT WHICH THE MICROCOMPUTER CAN PROCESS
AUTODIN TRAFFIC.

    THIS PROGRAM RUNS ON THE "BARE MACHINE", THAT IS,
IT DOES NOT REQUIRE A RESIDENT OPERATING SYSTEM; RATHER,
IT CONTAINS ITS OWN OPERATING SYSTEM WHICH PERMITS MULTI-
PROCESSING OF THE INDEPENDENT PROCESSES WHICH MAKE UP THE
TOTAL AUTODIN PROGRAM.  PROCESS SCHEDULING AND DEVICE MANAGE-
MENT ARE PERFORMED AT THE HIGHEST LEVEL OF THE PROGRAM.
THE NINE MAJOR PROCESSES WHICH ARE SCHEDULED ARE:

    1.  POLL$USAR
    2.  RECEIVE$LOGIC
    3.  WRITER
    4.  OPERATOR$INPUT
    5.  OPERATOR$OUTPUT
    6.  TRANSMIT$LOGIC
    7.  TRANSMITTER
    8.  POLL$DEVICES
    9.  POLL$USAT
```

```
    IT IS IMPORTANT TO NOTE THAT EACH PROCESS IS PERMIT-                    AUT00490
TED TO RUN FOR ONE BYTE AT A TIME; CONSEQUENTLY, SIMULTANEOUS              AUT00500
TRANSMISSION, RECEPTION, INPUT, AND OUTPUT MAY TAKE PLACE.                 AUT00510
                                                                           AUT00520
                                                                           AUT00530
    WHEN READING THE PROGRAM, BEGIN AT THE HIGHEST LEVEL,                  AUT00540
LEVEL 1, AND WORK THROUGH THE LEVELS 1, 2, 3, ...ETC.,                     AUT00550
SINCE THE HIGHER LEVEL PROCEDURES INVOKE THOSE OF LOWER                    AUT00560
LEVEL (AND LARGER NUMBER). ALL PROCEDURES ARE LABELLED                     AUT00570
WITH THEIR LEVEL NUMBER.  THE HIGHEST LEVEL, LEVEL 1,                      AUT00580
IS LOCATED AT THE BOTTOM OF THE PROGRAM LISTING.                           AUT00590
                                                                           AUT00600
                                                                           AUT00610
    FINALLY, EXTENSIVE USE OF THE LITERAL SUBSTITUTION CAPABILITY          AUT00620
OF PLM HAS BEEN MADE.  THIS PERMITS THE PROGRAM TO BE READ IN              AUT00630
ALGORITHMIC ENGLISH FORM BY THOSE WHO ARE INTERESTED IN                    AUT00640
ITS LOGICAL STRUCTURE.   THOSE INTERESTED IN BIT MANIPULATIONS            AUT00650
AND OTHER DETAILS MAY STUDY THE LITERAL DECLARATIONS.    */                AUT00660
                                                                           AUT00670
                                                                           AUT00680
                                                                           AUT00690
    /*   **********   LITERAL DECLARATIONS   **********   */               AUT00700
                                                                           AUT00710
DECLARE                                                                    AUT00720
    /* THE FOLLOWING LITERALS ARE USED FOR READABILITY */                  AUT00730
                                                                           AUT00740
    TRUE LITERALLY '1',                                                    AUT00750
    FALSE LITERALLY '0',                                                   AUT00760
    SPACE LITERALLY '20H',                                                 AUT00770
    FOREVER LITERALLY 'WHILE TRUE',                                        AUT00780
    NOT$READY LITERALLY '0',                                               AUT00790
    READY LITERALLY '1',                                                   AUT00800
    BOOT LITERALLY '0',                                                    AUT00810
    SCHEDULED LITERALLY '0',                                               AUT00820
    DEVICE$READY LITERALLY '1',                                            AUT00830
    LF LITERALLY '10',                                                     AUT00840
    CR LITERALLY '13',                                                     AUT00850
    EXPIRED LITERALLY '> 600',                                             AUT00860
    EVEN LITERALLY ' ',                                                    AUT00870
    LOW$7$MASK LITERALLY '7FH',    /* USED TO REMOVE PARITY BITS    */     AUT00880
    PARITY$MASK LITERALLY '80H',   /* USED TO INSERT PARITY BITS    */     AUT00890
    TTY LITERALLY '0',             /* USED FOR TELETYPE PORT        */     AUT00900
    MAG$TAPE LITERALLY '2',        /* USED FOR MAG TAPE I/O PORT    */     AUT00910
    CARD$PUNCH LITERALLY '3',      /* USED FOR CARD PUNCH I/O PORT */     AUT00920
    TTY$INFO LITERALLY '0',        /* USED FOR TTY I/O PORT         */     AUT00930
    TTY$STATUS LITERALLY '1',      /* USED FOR TTY STATUS PORT      */     AUT00940
    CRT$INFO LITERALLY '0',        /* USED FOR CRT I/O PORT         */     AUT00950
    CRT$STATUS LITERALLY '1',      /* USED FOR CRT STATUS PORT      */     AUT00960
```

```
    USART$INFO LITERALLY '4',      /* USED FOR USART I/O PORT       */   AUT00970
    USART$STATUS LITERALLY '5',    /* USED FOR USART STATUS PORT    */   AUT00980
    NOT$SELECTED LITERALLY '255', /* DENOTES NO DEVICE SELECTED    */   AUT00990
    BELL LITERALLY '07H',          /* ALERTS THE HUMAN OPERATOR     */   AUT01000
                                                                         AUT01010
    /* THE FOLLOWING LITERAL DECLARATIONS REPRESENT THE CONTROL         AUT01020
       AND FRAMING CHARACTERS USED IN AUTODIN   */                       AUT01030
                                                                         AUT01040
    SOH LITERALLY '81H',       /* START OF HEADER LINE  BLOCK       */  AUT01050
    STX LITERALLY '82H',       /* START OF INTERMEDIATE LINE BLOCK  */  AUT01060
    ETX LITERALLY '03H',       /* END OF LAST LINE BLOCK            */  AUT01070
    ACK1 LITERALLY '06H',      /* ACKNOWLEDGEMENT ONE               */  AUT01080
    INV LITERALLY '87H',       /* INVALID ACK OR NAK RECEIVED       */  AUT01090
    REP LITERALLY '11H',       /* REQUEST FOR ANSWER                */  AUT01100
    RM LITERALLY '12H',        /* REJECT YOUR MESSAGE               */  AUT01110
    NAK LITERALLY '95H',       /* NEGATIVE ACKNOWLEDGEMENT          */  AUT01120
    SYN LITERALLY '96H',       /* SYNCHRONOUS IDLE PATTERN          */  AUT01130
    ETB LITERALLY '17H',       /* END OF INTERMEDIATE LINE BLOCK    */  AUT01140
    CAN LITERALLY '18H',       /* CANCEL THIS MESSAGE               */  AUT01150
    EM LITERALLY '99H',        /* END OF TEXT MARKER                */  AUT01160
    MC LITERALLY '9AH',        /* MODE CHANGE                       */  AUT01170
    ACK2 LITERALLY '9CH',      /* ACKNOWLEDGEMENT TWO               */  AUT01180
    WBT LITERALLY '1EH',       /* WAIT REQUEST                      */  AUT01190
    DEL LITERALLY '0FFH';      /* DELETED FRAMING CHARACTER         */  AUT01200
                                                                         AUT01210
                                                                         AUT01220
    /*   **********   GLOBAL VARIABLE DECLARATIONS   **********   */    AUT01230
                                                                         AUT01240
                                                                         AUT01250
DECLARE                                                                  AUT01260
                                                                         AUT01270
    /* THE FOLLOWING VECTORS ARE USED BY THE EXECUTIVE                  AUT01280
       TO SCHEDULE THE DESCRIBED PROCESSES    */                        AUT01290
                                                                         AUT01300
    RECEIVE$LOGIC$PROCESS (2) BYTE,                                      AUT01310
    WRITER$PROCESS (2) BYTE,                                             AUT01320
    OPERATOR$INPUT$PROCESS (2) BYTE,                                     AUT01330
    OPERATOR$OUTPUT$PROCESS (2) BYTE,                                    AUT01340
    TRANSMIT$LOGIC$PROCESS (2) BYTE,                                     AUT01350
    TRANSMITTER$PROCESS (2) BYTE;                                        AUT01360
                                                                         AUT01370
                                                                         AUT01380
    /* GLOBAL FLAG AND VARIABLE DECLARATIONS */                          AUT01390
                                                                         AUT01400
DECLARE                                                                  AUT01410
                                                                         AUT01420
    USAR$CHECK BYTE,     /* USED TO DETECT IF THE USAR HAS GONE DEAD  */ AUT01430
    RCV$BYTE BYTE,       /* THE BYTE JUST RECEIVED FROM THE USAR      */ AUT01440
```

```
RCV$STATE BYTE,          /* THE STATE OF THE RECEIVER LOGIC PROCESS    */ AUT01450
RCV$EP BYTE,             /* THE BLOCK PARITY OF THE  ECEIVED BLOCK     */ AUT01460
SYN$CTR BYTE,            /* COUNTS THE NUMBER OF SYN RECEIVED IN A ROW*/ AUT01470
PREV$RCV$BYTE BYTE,      /* SAVES THE PREVIOUSLY RECEIVED BYTE         */ AUT01480
MC$FLAG BYTE,            /* TRUE=DATA FORMAT MODE CHANGE; FALSE O.W.   */ AUT01490
WBT$CTR BYTE,            /* COUNTS NUMBER OF WBT RECEIVED IN A ROW     */ AUT01500
NAK$CTR BYTE,            /* COUNTS NUMBER OF NAK RECEIVED IN A ROW     */ AUT01510
CHECK$RCV$TIMER BYTE,    /* TRUE MEANS CHECK THE RECEIVE TIMER         */ AUT01520
RCV$SYNCH$TIMER ADDRESS, /*IF TIMER GOES OFF, SYNCH HAS BEEN LOST*/      AUT01530
RCV$MID$MSG BYTE,        /* TRUE WHEN RCVR IS IN THE MIDDLE OF A MSG   */ AUT01540
RCV$TXT$CTR BYTE,        /* COUNTS THE TEXT FOR INCOMING BLOCKS        */ AUT01550
RCV$ERROR BYTE,          /* TRUE IF AN ERROR HAS BEEN FOUND IN BLOCK   */ AUT01560
PREV$RCV$STATE BYTE,     /* SAVES THE PREVIOUS STATE OF THE RCVR       */ AUT01570
RCV$ACK$1$2 BYTE,        /* 1=ACK1 TO BE RCVD NEXT; 2=ACK2 NEXT        */ AUT01580
BLOCK$START$STATE BYTE,  /* RCVR STATE AT BEGINNING OF BLOCK           */ AUT01590
RCV$REP$CTR BYTE,        /*  COUNTS NUMBER OF REP RECEIVED IN A ROW    */ AUT01600
OUTPUT$DEVICE BYTE,      /* RECEIVER SELECTS BASED ON RCVD SEL         */ AUT01610
STILL$WRITING BYTE,      /* TRUE MEANS WRITER IS STILL WRITING         */ AUT01620
                                                                          AUT01630
XMT$STATE BYTE,          /* THE STATE OF THE TRANSMIT LOGIC PROCESS    */ AUT01640
XMT$BYTE BYTE,           /* THE BYTE JUST READ IN AND TO BE XMTTED     */ AUT01650
XMT$BP BYTE,             /* KEEPS TRACK OF BLOCK PARITY FOR XMTR       */ AUT01660
XMT$TXT$CTR BYTE,        /* COUNTS TEXT FOR OUTGOING LINE BLOCKS       */ AUT01670
XMT$WAIT BYTE,           /* TRUE= XMTR WAITING FOR ACK'S               */ AUT01680
XMT$ACK$1$2 BYTE,        /* 1= SEND ACK1 NEXT, 2= SEND ACK2 NEXT       */ AUT01690
XMT$ANS$TIMER ADDRESS,   /* IF EXPIRED, ANSWER NOT RCVD SOON ENOUGH*/     AUT01700
CHECK$ANS$TIMER BYTE,    /* TRUE MEANS TO CHECK THE XMT TIMER          */ AUT01710
XMT$REP$CTR  BYTE,       /* KEEPS TRACK OF  REPS SENT IN A ROW         */ AUT01720
INPUT$DEVICE BYTE,       /* READ THIS DEVICE; TRANSMIT IT'S DATA       */ AUT01730
SENDING BYTE,            /* TRUE MEANS WE'RE SENDING A MSG NOW         */ AUT01740
CAN$FLAG BYTE,           /* TRUE MEANS WE'RE CANCELLING XMTD MSG       */ AUT01750
AWAITING$ACK BYTE,       /* TRUE=WAITING FOR LAST BLOCK TO BE ACKD     */ AUT01760
                                                                          AUT01770
OUTPUT$STATE BYTE,       /* USED TO PERFORM OPR OUTPUTS                */ AUT01780
ALARM$MSG BYTE,          /* USED TO SEND ONE LTR ALARM TO OPR          */ AUT01790
EOM$STATE BYTE,          /* USED TO CHECK FOR END OF MSG 'EOM'         */ AUT01800
LF$CTR BYTE,             /* USED TO COUNT LINE FEEDS (LOOKS FOR 8)     */ AUT01810
N$CTR BYTE,              /* USED TO COUNT N'S (LOOKS FOR 4 IN A ROW)   */ AUT01820
VIRTUAL BYTE,            /* TRUE MEANS READ/WRITE IN CORE; FALSE MEANS   AUT01830
                            READ/WRITE ON ACTUAL PERIPHERAL DEVICES    */ AUT01840
                                                                          AUT01850
RW$BUFFER$1 (81) BYTE,     /* UTILITY BUFFER FOR RECEIVE LOGIC AND       AUT01860
                              WRITER PROCESS                           */ AUT01870
RW$BUFFER$2 (81) BYTE,     /* SAME PURPOSE AS RW$BUFFER$1              */ AUT01880
RCV$BUFFER$ADDR ADDRESS,   /* SETS ADDR OF THE CURRENT RCV BUFFER      */ AUT01890
(RCV$BUFFER BASED RCV$BUFFER$ADDR) (81) BYTE,                             AUT01900
                           /* CURRENT BUFFER USED BY RCV LOGIC         */ AUT01910
WRITE$BUFFER$ADDR ADDRESS,                                                AUT01920
```

```
                                /* ADDRESS OF CURRENT WRITE BUFFER      */ AUT01930
    (WRITE$BUFFER BASED WRITE$BUFFER$ADDR) (81) BYTE,                      AUT01940
                                /* CURRENT BUFFER FOR WRITE PROCESS     */ AUT01950
    W$BUFFER$PTR BYTE,          /* POINTER FOR WRITE$BUFFER             */ AUT01960
                                                                           AUT01970
    T$BUFFER$1 (85) BYTE,       /* TWO UTILITY BUFFERS USED FOR XMT     */ AUT01980
    T$BUFFER$2 (85) BYTE,       /* AND TRANSMITTER PROCESSES            */ AUT01990
    XMT$LOGIC$BUFFER$ADDR ADDRESS,                                         AUT02000
                                /* SETS ADDR OF CURRENT XMT LOGIC BUFFER*/ AUT02010
    (XMT$LOGIC$BUFFER BASED XMT$LOGIC$BUFFER$ADDR) (85) BYTE,              AUT02020
                                /* CURRENT BUFFER USED BY XMT LOGIC     */ AUT02030
    TRANSMIT$BUFFER$ADDR ADDRESS,                                          AUT02040
                                /* SETS ADDR OF CURRENT TRANSMIT BUFFER */ AUT02050
    (TRANSMIT$BUFFER BASED TRANSMIT$BUFFER$ADDR) (85) BYTE,                AUT02060
                                /* THE CURRENT TRANSMIT PROCESS BUFFER  */ AUT02070
    T$BUFFER$PTR BYTE,          /* POINTER FOR TRANSMIT BUFFER          */ AUT02080
    T$BUFFER$LENGTH BYTE,       /* CURRENT LENGTH OF TRANSMIT BUFFER    */ AUT02090
                                                                           AUT02100
    CONCHAR$BUFFER (8) BYTE, /* BUFFER FOR THE CONTROL CHARS TO SEND */    AUT02110
    CC$PTR1 BYTE,               /* POINTER FOR CONTROL CHAR BUFFER      */ AUT02120
    CC$PTR2 BYTE,               /* POINTER FOR CONTROL CHAR BUFFER      */ AUT02130
                                                                           AUT02140
    /* THE FOLLOWING FOUR VECTORS ARE USED TO STORE RECEIVE               AUT02150
        LOGIC TABLE LOOKUP ACTIONS*/                                       AUT02160
                                                                           AUT02170
    TABLE$1 DATA(0,1,2,2,2,2,3,3,3,3,2,2,3,4,2,3,2,2,2,3,2,3,2),          AUT02180
    TABLE$2 DATA(0,2,5,2,2,2,3,3,3,3,2,2,3,4,2,3,2,2,2,3,2,3,2),          AUT02190
    TABLE$3 DATA(0,6,6,6,6,6,3,3,3,3,6,6,3,6,6,3,7,8,6,3,6,3,9),          AUT02200
    TABLE$4 DATA(0,10,10,11,10,10,3,3,3,3,10,10,3,12,13,3,10,10,          AUT02210
                 10,3,10,3,10),                                            AUT02220
                                                                           AUT02230
    /* THE FOLLOWING TWO VECTORS ARE USED TO STORE THE CORRECT            AUT02240
         'SEL' CHARACTERS SO THAT THE TRANSMIT LOGIC CAN INSERT            AUT02250
         THE CORRECT 'SEL' CHARACTER BASED ON THE LMF CHARACTER            AUT02260
         OF THE MESSAGE BEING TRANSMITTED  */                              AUT02270
                                                                           AUT02280
    SEL$LOOKUP$1 DATA('A','B','D','C',0,'H','A',0,'B'),                    AUT02290
    SEL$LOOKUP$2 DATA('A','A','C','H');                                    AUT02300
                                                                           AUT02310
                                                                           AUT02320
                                                                           AUT02330
    /* DECLARATIONS FOR TEST AND DEBUG */                                  AUT02340
                                                                           AUT02350
                                                                           AUT02360
DECLARE                                                                    AUT02370
                                                                           AUT02380
     /* THE FOLLOWING DECLARATIONS ARE FOR ODD PARITY TEXT                 AUT02390
        CHARACTERS WHICH ARE USED IN A TEST MESSAGE        */              AUT02400
```

```
OPA  LITERALLY '0C1H',                                                  AUT02410
OPB  LITERALLY '0C2H',                                                  AUT02420
OPC  LITERALLY '043H',                                                  AUT02430
OPD  LITERALLY '0C4H',                                                  AUT02440
CPE  LITERALLY '045H',                                                  AUT02450
OPF  LITERALLY '046H',                                                  AUT02460
OPG  LITERALLY '0C7H',                                                  AUT02470
OPH  LITERALLY '0C8H',                                                  AUT02480
CPI  LITERALLY '049H',                                                  AUT02490
OPJ  LITERALLY '04AH',                                                  AUT02500
CPK  LITERALLY '0CBH',                                                  AUT02510
OPL  LITERALLY '04CH',                                                  AUT02520
CPM  LITERALLY '0CDH',                                                  AUT02530
CPN  LITERALLY '0CEH',                                                  AUT02540
OPO  LITERALLY '04FH',                                                  AUT02550
OPP  LITERALLY '0D0H',                                                  AUT02560
OPQ  LITERALLY '051H',                                                  AUT02570
OPR  LITERALLY '052H',                                                  AUT02580
OPS  LITERALLY '0D3H',                                                  AUT02590
CPT  LITERALLY '054H',                                                  AUT02600
OPU  LITERALLY '0D5H',                                                  AUT02610
OPV  LITERALLY '0D6H',                                                  AUT02620
OPW  LITERALLY '057H',                                                  AUT02630
CPX  LITERALLY '058H',                                                  AUT02640
OPY  LITERALLY '0D9H',                                                  AUT02650
OPZ  LITERALLY '0DAH',                                                  AUT02660
OP0  LITERALLY '0B0H',                                                  AUT02670
OP1  LITERALLY '031H',                                                  AUT02680
CP2  LITERALLY '032H',                                                  AUT02690
OP3  LITERALLY '0B3H',                                                  AUT02700
CP4  LITERALLY '034H',                                                  AUT02710
OP5  LITERALLY '0B5H',                                                  AUT02720
OP6  LITERALLY '0B6H',                                                  AUT02730
CP7  LITERALLY '037H',                                                  AUT02740
OP8  LITERALLY '038H',                                                  AUT02750
OP9  LITERALLY '0B9H',                                                  AUT02760
CPDA LITERALLY '0ADH',                                                  AUT02770
OPCR LITERALLY '00DH',                                                  AUT02780
CPLF LITERALLY '08AH',                                                  AUT02790
OPSI LITERALLY '02FH',                                                  AUT02800
OPPD LITERALLY '023H',                                                  AUT02810
OPPE LITERALLY '0AEH',                                                  AUT02820
OPSE LITERALLY '020H',                                                  AUT02830
EPA  LITERALLY '041H';                                                  AUT02840
                                                                        AUT02850
                                                                        AUT02860
                                                                        AUT02870
 /* TEST$MSG IS USED WHEN A TEST MESSAGE IS DESIRED FOR                 AUT02880
```

```
        TESTING THE RECEIVER AUTOMATON BY ITSELF                    */
DECLARE TEST$MSG DATA(SYN,
SYN, SYN, SYN, SYN, SYN, SYN, SYN, SYN, SYN, SYN,
SYN, SYN, SYN, SYN, SYN, SYN, SYN, SYN, SOH, EPA,
OPR, OPT, OPT, OPU, OPZ, OPY, OPU, OPW, OPSP,OPR,
OPU, OPW, OPJ, OPA, OPG, OPC, OP0, OP0, OP0, OP0,
OPSP,OP2, OP1, OP2, OP2, OP2, OP0, OP0, OPDA,OPU,
OPU, OPU, OPU, OPDA,OPDA,OPR, OPU, OPW, OPJ, OPA,
OPG, OPC, OPPE,OPCR,OPLF,OPZ, OPN, OPR, OPSP,OPU,
OPU, OPU, OPU, OPU, OPCR,OPLF,OPR, OPSP,OP3, OP0,
OP2, OP2, OP2, OP0, OP0, OPZ, OPSP,OPJ, OPU, OPL,
OPSP,OP7, OP6, OPCR,OPLF,OPF, OPM, OPSP,OPN,OPT,
ETB,   0, SYN, SYN, SYN, SYN, SYN, SYN, SYN, SYN, SYN,
SYN, SYN, SYN, SYN, SYN, SYN, SYN, SYN, STX, DEI,
OPC, OPC, OPSP,OPM, OPO, OPN, OPT, OPE, OPR, OPE,
OPY, OPCR,OPLF,OPT, OPO, OPSP,OPN, OPT, OPC, OPC,
OPSP,OPM, OPO, OPN, OPT, OPE, OPR, OPE, OPY, OPCR,
OPLF,OPE, OPT, OPCR,OPLF,OPU, OPN, OPC, OPL, OPA,
OPS, OPSL,OPSL,OPN, OP0, OP0, OP0, OP0, OP0, OPSP,
OPSL,OPCR,OPLF,OPN, OPO, OPW, OPSP,OPI, OPS, OPSP,
OPT, OPH, OPE, OPSP,OPT, OPI, OPM, OPE, OPSP,OPF,
OPO, OPR, OPSP,OPA, OPL, OPL, OPSP,OPG, OPO, OPO,
ETB,   0, SYN, SYN, SYN, SYN, SYN, SYN, SYN, SYN,
SYN, SYN, SYN, SYN, SYN, SYN, SYN, SYN, STX, DEI,
OPD, OPSP,OPM, OPE, OPN, OPSP,OPT, OPO, OPSP,OPC,
OPO, OPM, OPE, OPSP,OPT, OPO, OPSP, OPT, OPH,OPE,
OPSP,OPA, OPI, OPD, OPSP,OPO, OPF, OPT, OPH, OPE,
OPI, OPR, OPSP,OPC, OPO, OPU, OPN, OPT, OPR, OPY,
OPPE,OPCR,OPLF,OPB, OPT, OPCR,OPLF,OPPD,OP0, OP0,
OP0, OP0, OPCR,OPLF,OPLF,OPLF,OPLF,OPLF,OPLF,OPLF,
OPLF,OPN, OPN, OPN, OPN,  EM, ETX,   0, SYN, SYN,
SYN, SYN, SYN, SYN, SYN, SYN, SYN, SYN, SYN, SYN);

     /* TEST$TAPE IS USED FOR TESTING THE TRANSMITTER - IT
        PROVIDES A VIRTUAL PAPER TAPE FOR THE TRANSMIT LOGIC */

DECLARE TEST$TAPE DATA(' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ', ' ',
'R', 'I', 'T', 'U', 'Z', 'Y', 'U', 'W', ' ', 'R',
'R', 'U', 'W', 'J', 'A', 'G', 'C', '0', '0', '0',
'0', '2', '1', '2', '2', '2', '0', '0', '-', 'U',
'U', 'U', 'U', '-', '-', 'R', 'U', 'W', 'J', 'A',
'G', 'C', 0DH,0AH, 'A', 'U', 'T', 'O', 'D', 'I',
'N', ' ', 'T', 'E', 'S', 'T', ' ', 'M', 'E', 'S',
'S', 'A', 'G', 'E', ' ', 'A', 'B', 'C', 'D', 'E',
'F', 'G', 'H', 'I', 'J', 'K', 'L', 'M', 'N', 'O',
'P', 'Q', 'R', 'S', 'T', 'U', 'V', 'W', 'X', 'Y',
'Z', '0', '1', '2',0AH,0AH,0AH,0AH,0AH,0AH,
```

```
0AH,0AH,'N','N','N','N');                                          AUT03370
DECLARE R$TEST$PTR ADDRESS;                                        AUT03380
                                                                   AUT03390
    /* PAPER IS A VECTOR WHICH SERVES AS A VIRTUAL PAPER */        AUT03400
                                                                   AUT03410
DECLARE PAPER(401) BYTE;                                           AUT03420
                                                                   AUT03430
    /* THE FOLLOWING DECLARATIONS ARE FOR POINTERS WHICH          AUT03440
       REQUIRED FOR THE VIRTUAL BUFFERS PREVIOUSLY DECLARED */     AUT03450
                                                                   AUT03460
DECLARE S$TEST$PTR ADDRESS;                                        AUT03470
DECLARE T$TEST$PTR BYTE;                                           AUT03480
DECLARE PAPER$PTR ADDRESS;                                         AUT03490
DECLARE BYTE$COUNTER ADDRESS;                                      AUT03500
DECLARE TEST$COUNTER BYTE;                                         AUT03510
DECLARE DEBUG$FLAG BYTE;                                           AUT03520
                                                                   AUT03530
                                                                   AUT03540
    /* END OF TEST/DEBUG DECLARATIONS */                           AUT03550
                                                                   AUT03560
                                                                   AUT03570
                                                                   AUT03580
    /* ********************************************************* */AUT03590
    /* *****     GLOBAL UTILITY PROCEDURES    ************** */   AUT03600
                                                                   AUT03610
CRT$OUT:PROCEDURE(CHARACTER);                                      AUT03620
    /* LEVEL 5 PROCEDURE */                                        AUT03630
                                                                   AUT03640
    /* SENDS ONE CHARACTER TO THE CONSOLE DEVICE */                AUT03650
                                                                   AUT03660
     DECLARE CHARACTER BYTE;                                       AUT03670
                                                                   AUT03680
    IF NOT VIRTUAL THEN                                            AUT03690
        DO;                                                        AUT03700
            DO WHILE ROR(INPUT(CRT$STATUS),2);                     AUT03710
            END;  /* WAIT UNTIL CONSOLE IS READY */                AUT03720
            OUTPUT(CRT$INFO)=NOT CHARACTER;                        AUT03730
        END;                                                       AUT03740
                                                                   AUT03750
END CRT$OUT;                                                       AUT03760
                                                                   AUT03770
CRT$IN: PROCEDURE BYTE;                                            AUT03780
                                                                   AUT03790
    /* LEVEL 5 PROCEDURE */                                        AUT03800
                                                                   AUT03810
    /* INPUTS ONE BYTE FROM THE CONSOLE DEVICE */                  AUT03820
                                                                   AUT03830
    RETURN (NOT INPUT(CRT$INFO)) AND LOW$7$MASK;                   AUT03840
```

```
END CRT$IN;

SEND$CC: PROCEDURE(CONTROL$CHAR);

    /* LEVEL 5 PROCEDURE */

    /* PUTS TWO CONTROL CHARACTERS IN THE BUFFER  TO BE SENT */

    DECLARE CONTROL$CHAR BYTE;

    CONCHAR$BUFFER(CC$PTR2),CONCHAR$BUFFER(CC$PTR2 + 1)=CONTROL$CHAR;
    IF (CC$PTR2:=CC$PTR2 + 2) > 7 THEN
         CC$PTR2=0;

END SEND$CC;

INITIALIZE$RCVR: PROCEDURE;

    /* LEVEL 5 PROCEDURE */

    /* INITIALIZES THE RECEIVE LOGIC AND WRITER PROCESSES */

    CHECK$RCV$TIMER=TRUE;
    MC$FLAG, RCV$MID$MSG, RCV$ERROR, STILL$WRITING=FALSE;

    XMT$ACK$1$2, W$BUFFER$PTR, PREV$RCV$STATE, BLOCK$START$STATE=1;

    RCV$BP, SYN$CTR, WBT$CTR, NAK$CTR, RCV$SYNCH$TIMER,
    RCV$TXT$CTR, RCV$REP$CTR, RW$BUFFER$1, RW$BUFFER$2=0;

    OUTPUT$DEVICE=NOT$SELECTED;

    RCV$BUFFER$ADDR,WRITE$BUFFER$ADDR=.RW$BUFFER$1;

    RCV$STATE=5;

    RECEIVE$LOGIC$PROCESS(SCHEDULED)=TRUE;

    WRITER$PROCESS(SCHEDULED)=FALSE;

    IF NOT VIRTUAL THEN
         DO;
              RCV$STATE=1;
              RECEIVE$LOGIC$PROCESS(SCHEDULED)=FALSE;
         END;

END INITIALIZE$RCVR;
```

AUT03850 AUT03860 AUT03870 AUT03880 AUT03890 AUT03900 AUT03910 AUT03920 AUT03930 AUT03940 AUT03950 AUT03960 AUT03970 AUT03980 AUT03990 AUT04000 AUT04010 AUT04020 AUT04030 AUT04040 AUT04050 AUT04060 AUT04070 AUT04080 AUT04090 AUT04100 AUT04110 AUT04120 AUT04130 AUT04140 AUT04150 AUT04160 AUT04170 AUT04180 AUT04190 AUT04200 AUT04210 AUT04220 AUT04230 AUT04240 AUT04250 AUT04260 AUT04270 AUT04280 AUT04290 AUT04300 AUT04310 AUT04320

```
INITIALIZE$XMTR: PROCEDURE;                                              AUT04330
                                                                         AUT04340
    /* LEVEL 5 PROCEDURE */                                              AUT04350
                                                                         AUT04360
    /* INITIALIZES THE TRANSMIT LOGIC AND TRANSMITTER PROCESSES */       AUT04370
                                                                         AUT04380
    XMT$WAIT, CHECK$ANS$TIMER, CAN$FLAG, SENDING,                        AUT04390
    AWAITING$ACK=FALSE;                                                  AUT04400
                                                                         AUT04410
    RCV$ACK$1$2, XMT$STATE, T$BUFFER$PTR, EOM$STATE=1;                   AUT04420
                                                                         AUT04430
    XMT$BP, XMT$ANS$TIMER, XMT$REP$CTR, T$BUFFER$1,                      AUT04440
    T$BUFFER$2, LF$CTR, N$CTR=0;                                         AUT04450
                                                                         AUT04460
    INPUT$DEVICE=NOT$SELECTED;                                           AUT04470
                                                                         AUT04480
    XMT$LOGIC$BUFFER$ADDR=.T$BUFFER$2;                                   AUT04490
                                                                         AUT04500
    TRANSMIT$BUFFER$ADDR=.T$BUFFER$1;                                    AUT04510
                                                                         AUT04520
    TRANSMIT$LOGIC$PROCESS(DEVICE$READY)=FALSE;                          AUT04530
                                                                         AUT04540
END INITIALIZE$XMTR;                                                     AUT04550
                                                                         AUT04560
ALARM: PROCEDURE (CONDITION);                                            AUT04570
                                                                         AUT04580
     /* LEVEL 5 PROCEDURE */                                             AUT04590
                                                                         AUT04600
    /* SENDS AN APPROPRIATE ALARM MESSAGE BY SCHEDULING THE             AUT04610
    OPERATOR OUTPUT PROCESS. THE ALARM CONDITION IS COMMUNICATED         AUT04620
    VIA GLOBAL VARIABLE 'ALARM$MSG'.  RE-INITIALIZATION OF               AUT04630
    OF THE RECEIVER OR TRANSMITTER IS PERFORMED WHEN NECESSARY. */       AUT04640
                                                                         AUT04650
    DECLARE CONDITION BYTE;                                              AUT04660
    DO CASE CONDITION;                                                   AUT04670
                                                                         AUT04680
        ; /* CASE 0 NOT USED */                                          AUT04690
                                                                         AUT04700
        DO;  /* CASE 1 -- 3 NAK'S RECEIVED IN A ROW */                   AUT04710
             CALL SEND$CC(CAN);                                          AUT04720
             ALARM$MSG='N';                                              AUT04730
             CALL INITIALIZE$XMTR;                                       AUT04740
        END;                                                             AUT04750
                                                                         AUT04760
        DO;  /* CASE 2 - - 3 WBT'S RCVD IN A ROW */                      AUT04770
             CALL SEND$CC(CAN);                                          AUT04780
             ALARM$MSG='W';                                              AUT04790
                                                                         AUT04800
```

```
                CALL INITIALIZE$XMTR;                                   AUT04810
            END;                                                        AUT04820
                                                                        AUT04830
            DO;  /* CASE 3 -- CAN RCVD FROM THE ASC */                  AUT04840
                 ALARM$MSG='C';                                         AUT04850
                 CALL INITIALIZE$RCVR;                                  AUT04860
            END;                                                        AUT04870
                                                                        AUT04880
                 /* CASE 4 -- REP RECEIVED BETWEEN MSGS */              AUT04890
            ALARM$MSG='R';                                              AUT04900
                                                                        AUT04910
            DO;  /* CASE 5 -- RM RECEIVED */                            AUT04920
                 CALL SEND$CC(CAN);                                     AUT04930
                 ALARM$MSG='M';                                         AUT04940
                 CALL INITIALIZE$RCVR;                                  AUT04950
            END;                                                        AUT04960
                                                                        AUT04970
                 /* CASE 6 -- INV RCVD */                               AUT04980
            ALARM$MSG='I';                                              AUT04990
                                                                        AUT05000
            DO;  /* CASE 7 -- 3 REP'S SENT IN A ROW */                  AUT05010
                 CALL SEND$CC(CAN);                                     AUT05020
                 ALARM$MSG='3';                                         AUT05030
                 CALL INITIALIZE$XMTR;                                  AUT05040
            END;                                                        AUT05050
                                                                        AUT05060
            DO;  /* CASE 8 -- INVALID LMF ON OUTGOING MSG */            AUT05070
                 ALARM$MSG='L';                                         AUT05080
                 CALL INITIALIZE$XMTR;                                  AUT05090
            END;                                                        AUT05100
                                                                        AUT05110
                 /* CASE 9 FLASH; SEND BELL AND F TO CONSOLE */         AUT05120
            ALARM$MSG='F';                                              AUT05130
                                                                        AUT05140
                 /* CASE 10 - LOSS OF SYNCH AND SYNC TIMER EXPD */      AUT05150
            ALARM$MSG='S';                                              AUT05160
                                                                        AUT05170
                 /* CASE 11 - USAR DEAD */                              AUT05180
            ALARM$MSG='D';                                              AUT05190
                                                                        AUT05200
        END; /* OF CASE CONDITION */                                    AUT05210
                                                                        AUT05220
        OUTPUT$STATE=1;                                                 AUT05230
        CPERATOR$OUTPUT$PROCESS(SCHEDULED)=TRUE;                        AUT05240
                                                                        AUT05250
    END ALARM;                                                          AUT05260
                                                                        AUT05270
    GET$RCVD$BYTE: PROCEDURE BYTE;                                      AUT05280
```

```
     /* LEVEL 5 PROCEDURE */

     RCV$SYNCH$TIMER=RCV$SYNCH$TIMER + 1;
     IF CHECK$ANS$TIMER THEN
          XMT$ANS$TIMER=XMT$ANS$TIMER + 1;
     ELSE XMT$ANS$TIMER=0;
     /* THESE TWO COUNTERS ARE BUMPED EACH TIME A BYTE IS RECEIVED.
     WHEN THEY GET OVER 600 IN VALUE, THE TIMERS HAVE EXPIRED.  */
     USAR$CHECK=0;   /* JUST GOT A BYTE; SO USAR IS GOOD   */


IF VIRTUAL THEN
     DO;
     BYTE$COUNTER=BYTE$COUNTER + 1;
     IF(R$TEST$PTR:=R$TEST$PTR + 1) > 300 THEN
          R$TEST$PTR=0;
     RETURN TEST$MSG(R$TEST$PTR);

     END;
ELSE RETURN NOT(INPUT(USART$INFO));

END GET$RCVD$BYTE;

WRITE$BYTE: PROCEDURE (CHARACTER);

     /* LEVEL 5 PROCEDURE */

     DECLARE CHARACTER BYTE;

IF VIRTUAL THEN
     DO;
     PAPER(PAPER$PTR)=CHARACTER;
     PAPER$PTR=PAPER$PTR + 1;
     IF PAPER$PTR > 400 THEN
          PAPER$PTR=0;

     END;
ELSE
     DO;
          IF OUTPUT$DEVICE=TTY THEN
               DO;
                    DO WHILE ROR(INPUT(TTY$STATUS),2);
                    END;      /* WAIT UNTIL TTY IS READY */
                    OUTPUT(TTY)=NOT CHARACTER;
               END;
          ELSE
               IF OUTPUT$DEVICE=MAG$TAPE THEN
```

AUT05290 AUT05300 AUT05310 AUT05320 AUT05330 AUT05340 AUT05350 AUT05360 AUT05370 AUT05380 AUT05390 AUT05400 AUT05410 AUT05420 AUT05430 AUT05440 AUT05450 AUT05460 AUT05470 AUT05480 AUT05490 AUT05500 AUT05510 AUT05520 AUT05530 AUT05540 AUT05550 AUT05560 AUT05570 AUT05580 AUT05590 AUT05600 AUT05610 AUT05620 AUT05630 AUT05640 AUT05650 AUT05660 AUT05670 AUT05680 AUT05690 AUT05700 AUT05710 AUT05720 AUT05730 AUT05740 AUT05750 AUT05760

```
                    OUTPUT(MAG$TAPE)=NOT CHARACTER;                    AUT05770
                ELSE OUTPUT(CARD$PUNCH)=NOT CHARACTER;                 AUT05780
    END;                                                               AUT05790
                                                                       AUT05800
END WRITE$BYTE;                                                        AUT05810
                                                                       AUT05820
SEND$BYTE: PROCEDURE (CHARACTER);                                      AUT05830
                                                                       AUT05840
     /* LEVEL 5 PROCEDURE */                                           AUT05850
                                                                       AUT05860
     DECLARE CHARACTER BYTE;                                           AUT05870
                                                                       AUT05880
                                                                       AUT05890
IF VIRTUAL THEN                                                        AUT05900
    DO;                                                                AUT05910
                                                                       AUT05920
    IF(S$TEST$PTR:=S$TEST$PTR + 1)>300 THEN                            AUT05930
        S$TEST$PTR=0;                                                  AUT05940
    TEST$MSG(S$TEST$PTR)=CHARACTER;                                    AUT05950
    END;                                                               AUT05960
ELSE                                                                   AUT05970
    OUTPUT(USART$INFO)=NOT CHARACTER;                                  AUT05980
                                                                       AUT05990
END SEND$BYTE;                                                         AUT06000
                                                                       AUT06010
                                                                       AUT06020
GET$BYTE$TO$XMT: PROCEDURE BYTE;                                       AUT06030
                                                                       AUT06040
     /* LEVEL 5 PROCEDURE */                                           AUT06050
                                                                       AUT06060
IF VIRTUAL THEN                                                        AUT06070
    DO;                                                                AUT06080
    IF(T$TEST$PTR:=T$TEST$PTR + 1) > 115 THEN                          AUT06090
        T$TEST$PTR=0;                                                  AUT06100
    RETURN TEST$TAPE(T$TEST$PTR);                                      AUT06110
    END;                                                               AUT06120
ELSE                                                                   AUT06130
    DO;                                                                AUT06140
        IF INPUT$DEVICE=TTY THEN                                       AUT06150
            DO;                                                        AUT06160
                IF INPUT(TTY$STATUS) THEN /* STROBE READER */         AUT06170
                    DO;                                                AUT06180
                        OUTPUT(TTY$STATUS)=1;                          AUT06190
                        OUTPUT(TTY$STATUS)=0;                          AUT06200
                    END;                                               AUT06210
                DO WHILE INPUT(TTY$STATUS);                            AUT06220
                END;  /* WAIT UNTIL TTY IS READY */                    AUT06230
                RETURN(NOT INPUT (TTY$INFO) AND LOW$7$MASK);           AUT06240
```

```
            END;                                                                AUT06250
        ELSE                                                                    AUT06260
            IF INPUT$DEVICE=MAG$TAPE THEN                                       AUT06270
                RETURN (NOT(INPUT(MAG$TAPE))AND LOW$7$MASK);                    AUT06280
            ELSE                                                                AUT06290
                RETURN (NOT(INPUT(CARD$PUNCH)) AND LOW$7$MASK);                 AUT06300
    END;                                                                        AUT06310
                                                                                AUT06320
END GET$BYTE$TO$XMT;                                                            AUT06330
                                                                                AUT06340
                                                                                AUT06350
    /*      ********************************************************    */    AUT06360
    /*       END OF THE LEVEL 5 UTILITY PROCEDURES                      */    AUT06370
                                                                                AUT06380
                                                                                AUT06390
    /*      ********************************************************    */    AUT06400
    /*     THE 10 MAJOR PROCESSES (LEVEL 2 PROCEDURES) ARE LISTED BELOW*/      AUT06410
                                                                                AUT06420
                                                                                AUT06430
INITIALIZE: PROCEDURE;                                                          AUT06440
    /* LEVEL 2 PROCEDURE */                                                     AUT06450
                                                                                AUT06460
    /* INITIALIZES EVERYTHING AT POWER - UP */                                  AUT06470
                                                                                AUT06480
    CALL INITIALIZE$RCVR;                                                       AUT06490
    CALL INITIALIZE$XMTR;                                                       AUT06500
    CC$PTR1,CC$PTR2, USAR$CHECK=0;                                              AUT06510
                                                                                AUT06520
    RECEIVE$LOGIC$PROCESS(DEVICE$READY)=FALSE;                                  AUT06530
    WRITER$PROCESS(DEVICE$READY)=FALSE;                                         AUT06540
    OPERATOR$INPUT$PROCESS(DEVICE$READY)=FALSE;                                 AUT06550
    OPERATOR$OUTPUT$PROCESS(SCHEDULED),                                         AUT06560
    OPERATOR$OUTPUT$PROCESS(DEVICE$READY)=FALSE;                                AUT06570
    TRANSMITTER$PROCESS(DEVICE$READY)=FALSE;                                    AUT06580
                                                                                AUT06590
    OUTPUT(TTY$STATUS)=1;   /* STROBE TTY READER */                             AUT06600
    OUTPUT(TTY$STATUS)=0;                                                       AUT06610
                                                                                AUT06620
    /* INITIALIZE TEST/DEBUG BUFFERS AND POINTERS, ETC. */                      AUT06630
                                                                                AUT06640
    R$TEST$PTR, BYTE$COUNTER, TEST$COUNTER=0;                                   AUT06650
                                                                                AUT06660
    /* INITIALIZE PAPER BUFFER */                                               AUT06670
                                                                                AUT06680
    DO PAPER$PTR=0 TO 400;                                                      AUT06690
        PAPER(PAPER$PTR)=0;                                                     AUT06700
    END;                                                                        AUT06710
    PAPER$PTR=0;                                                                AUT06720
```

```
    S$TEST$PTR=7;                                                    AUT06730
    T$TEST$PTR=0;                                                    AUT06740
     SENDING=TRUE;                                                   AUT06750
                                                                     AUT06760
    /* END OF TEST DEBUG INITIALIZATION */                           AUT06770
END INITIALIZE;                                                      AUT06780
                                                                     AUT06790
POLL$USAR: PROCEDURE;                                                AUT06800
                                                                     AUT06810
    /*   LEVEL 2 PROCEDURE   */                                      AUT06820
                                                                     AUT06830
    /* POLLS THE RECEIVE SIDE OF THE USART TO SEE IF A RECEIVED     AUT06840
         BYTE IS READY FOR PROCESSING.   */                          AUT06850
                                                                     AUT06860
                                                                     AUT06870
    IF INPUT(USART$STATUS) THEN /* USAR NOT READY */                 AUT06880
         RECEIVE$LOGIC$PROCESS(DEVICE$READY)=FALSE;                  AUT06890
    ELSE /* DEVICE IS READY */                                       AUT06900
         RECEIVE$LOGIC$PROCESS(DEVICE$READY)=TRUE;                   AUT06910
                                                                     AUT06920
                                                                     AUT06930
                                                                     AUT06940
IF VIRTUAL THEN                                                      AUT06950
    RECEIVE$LOGIC$PROCESS(DEVICE$READY)=TRUE;                        AUT06960
                                                                     AUT06970
END POLL$USAR;                                                       AUT06980
                                                                     AUT06990
RECEIVE$LOGIC: PROCEDURE;                                            AUT07000
                                                                     AUT07010
    /*   LEVEL 2 PROCEDURE   */                                      AUT07020
                                                                     AUT07030
    /*  EXAMINES INCOMING (RECEIVED) BYTES ONE AT A TIME.            AUT07040
        DETERMINES LOSS OR GAIN OF SYNCHRONOUS IDLE, CHECKS          AUT07050
        FOR ODD/EVEN PARITY, RECOGNIZES AND ACTS UPON CONTROL        AUT07060
        CHARACTERS, SCHEDULES THE TRANSMITTER PROCESS WHEN           AUT07070
        NECESSARY, PLACES TEST BYTES IN BUFFERS, SCHEDULES           AUT07080
        THE WRITER PROCESS WHEN THESE BUFFERS ARE READY FOR          AUT07090
        WRITING, SELECTS THE CORRECT WRITER (OUTPUT) DEVICE          AUT07100
        BY EXAMINING THE 'SEL' CHARACTER, SETS TIMERS AND            AUT07110
        LOGICAL FLAGS AND VARIABLES BASED UPON RECEIVED              AUT07120
        INFORMATION, SCHEDULES OPERATOR CONSOLE PROCESS              AUT07130
        WHENEVER ALARM CONDITIONS OCCUR, AND SCHEDULES ITSELF        AUT07140
        WHENEVER SYNCHRONIZATION MUST BE ACHIEVED.                   AUT07150
                                                                     AUT07160
       THE RECEIVE LOGIC PROCEDURE IS A 9-STATE AUTOMATON            AUT07170
       WHICH HAS THE FOLLOWING 9 STATES:                             AUT07180
                                                                     AUT07190
           1. WAITING FOR SOH.                                       AUT07200
```

```
            2. WAITING FOR STX.                                          AUT07210
            3. PROCESSING TEXT.                                          AUT07220
            4. WAITING FOR ETB.                                          AUT07230
            5. ATTEMPTING TO ACHIEVE SYNCHRONIZATION.                    AUT07240
            6. EXPECTING SEL.                                            AUT07250
            7. EXPECTING DEL.                                            AUT07260
            8. EXPECTING BLOCK PARITY.                                   AUT07270
            9. EXPECTING SECOND CHARACTER OF 2-CHARACTER CONTROL         AUT07280
               SEQUENCE.*/                                               AUT07290
                                                                         AUT07300
DECLARE                                                                  AUT07310
    LCOKUP BYTE,     /* USED TO INDEX ACTION LCOKUPS */                  AUT07320
    ACTION BYTE;       /* THE ACTION 1-13 PERFORMED */                   AUT07330
                                                                         AUT07340
PERFORM$ACTION: PROCEDURE;                                               AUT07350
                                                                         AUT07360
/* LEVEL 3 PROCEDURE */                                                  AUT07370
                                                                         AUT07380
/* PERFORMS ONE OF 13 ACTIONS FOR RECEIVER STATES 1-4 */                 AUT07390
                                                                         AUT07400
                                                                         AUT07410
    DO CASE ACTION;                                                      AUT07420
                                                                         AUT07430
        ;/* CASE 0 NOT USED */                                          AUT07440
                                                                         AUT07450
        DO;/* CASE 1: EXPECTED SOH, GOT SOH */                          AUT07460
            RCV$STATE=6;                                                 AUT07470
            RCV$ERROR, CHECK$RCV$TIMER=FALSE;                            AUT07480
           BLOCK$START$STATE=1;                                          AUT07490
        END;                                                             AUT07500
                                                                         AUT07510
            /* CASE 2: DIDN'T GET EXPECTED CHARACTER */                  AUT07520
        SYN$CTR=0;                                                       AUT07530
                                                                         AUT07540
        DO;/* CASE 3: FIRST OF 2-CHARACTER CONTROL SEQUENCE */          AUT07550
            SYN$CTR=0;                                                   AUT07560
            PREV$RCV$STATE=RCV$STATE;/* SAVE PREVIOUS STATE */           AUT07570
            PREV$RCV$BYTE=RCV$BYTE;/* SAVE PREVIOUS BYTE */             AUT07580
            RCV$STATE=9;                                                 AUT07590
        END;                                                             AUT07600
                                                                         AUT07610
        DO;/* CASE 4: SYN RECEIVED BETWEEN LINE BLOCKS */               AUT07620
            SYN$CTR=SYN$CTR + 1;                                         AUT07630
            IF SYN$CTR > 3 THEN                                          AUT07640
            SYN$CTR,RCV$SYNCH$TIMER=0;                                   AUT07650
        END;                                                             AUT07660
                                                                         AUT07670
        DO;/* CASE 5: EXPECTED STX, GOT STX */                          AUT07680
```

```
    CHECK$RCV$TIMER=FALSE;                                          AUT07690
    BLOCK$START$STATE=2;                                            AUT07700
    RCV$STATE=7;                                                    AUT07710
END;                                                                AUT07720
                                                                    AUT07730
DO;/* CASE 6: PROCESSING TEXT, INVALID CHAR RECEIVED */            AUT07740
    RCV$TXT$CTR=RCV$TXT$CTR + 1;                                    AUT07750
    RCV$ERROR=TRUE;                                                 AUT07760
    IF RCV$TXT$CTR > 80 THEN                                        AUT07770
    RCV$STATE=4;                                                    AUT07780
END;                                                                AUT07790
                                                                    AUT07800
DO;  /* CASE 7: PROCESSING TEXT, RECEIVE EM */                      AUT07810
    RCV$BP=RCV$BP + RCV$BYTE;                                       AUT07820
    RCV$STATE=4;                                                    AUT07830
END;                                                                AUT07840
                                                                    AUT07850
DO; /* CASE 8: PROCESSING TEXT, MC RECEIVED */                     AUT07860
    MC$FLAG=TRUE;                                                   AUT07870
    RCV$BP=RCV$BP + RCV$BYTE;                                       AUT07880
END;                                                                AUT07890
                                                                    AUT07900
DO;  /* CASE 9: PROCESSING TEXT, GET TEXT CHARACTER */              AUT07910
    RCV$BP=RCV$BP + RCV$BYTE;                                       AUT07920
    RCV$TXT$CTR=RCV$TXT$CTR + 1;                                    AUT07930
    RCV$BUFFER(RCV$TXT$CTR)=RCV$BYTE AND LOW$7$MASK;                AUT07940
    IF RCV$TXT$CTR=80 THEN RCV$STATE=4;                             AUT07950
    /* MASK REMOVES LEFTMOST PARITY BIT */                          AUT07960
END;                                                                AUT07970
                                                                    AUT07980
DO;  /* CASE 10: EXPECTING ETB/ETX, GET ERRONEOUS CHAR */           AUT07990
    RCV$ERROR=TRUE;                                                 AUT08000
    RCV$STATE=8;                                                    AUT08010
END;                                                                AUT08020
                                                                    AUT08030
DO;  /* CASE 11: EXPECTING ETB, GET ETX */                          AUT08040
    RCV$MID$MSG,MC$FLAG=FALSE;                                      AUT08050
    RCV$BP=RCV$BP + RCV$BYTE;                                       AUT08060
    RCV$STATE=8;                                                    AUT08070
END;                                                                AUT08080
                                                                    AUT08090
; /* CASE 12: GET SYN AT END OF BLOCK - DO NOTHING */               AUT08100
                                                                    AUT08110
DO; /* CASE 13: EXPECTING ETB/ETX, GET ETB */                       AUT08120
    RCV$BP=RCV$BP + RCV$BYTE;                                       AUT08130
    RCV$STATE=8;                                                    AUT08140
END;                                                                AUT08150
                                                                    AUT08160
```

```
    END;   /* OF CASE ACTION */                                          AUT08170
                                                                         AUT08180
END PERFORM$ACTION;                                                      AUT08190
                                                                         AUT08200
RCV$STATE$5: PROCEDURE;                                                  AUT08210
                                                                         AUT08220
     /* LEVEL 3 PROCEDURE */                                             AUT08230
                                                                         AUT08240
     /* WHEN THE RECEIVER IS IN STATE 5 IT IS ATTEMPTING                AUT08250
          TO ACHIEVE SYNCHRONIZATION */                                  AUT08260
                                                                         AUT08270
      RCV$SYNCH$TIMER=0;                                                 AUT08280
     SYN$CTR=0;                                                          AUT08290
     DO WHILE SYN$CTR < 4;                                               AUT08300
          DO WHILE NOT (RECEIVE$LOGIC$PROCESS                            AUT08310
               (DEVICE$READY));                                          AUT08320
               CALL POLL$USAR;                                           AUT08330
          END;                                                           AUT08340
          IF (RCV$BYTE:=GET$RCVD$BYTE)=SYN THEN                          AUT08350
               SYN$CTR=SYN$CTR + 1;                                      AUT08360
          ELSE SYN$CTR = 0;                                              AUT08370
           IF RCV$SYNCH$TIMER EXPIRED THEN CALL ALARM(10);               AUT08380
     END;                                                                AUT08390
                                                                         AUT08400
     RECEIVE$LOGIC$PROCESS(SCHEDULED)=FALSE;                             AUT08410
     RCV$SYNCH$TIMER=0;                                                  AUT08420
     CHECK$RCV$TIMER=TRUE;                                               AUT08430
     IF RCV$MID$MSG THEN RCV$STATE=2;                                    AUT08440
     ELSE RCV$STATE=1;                                                   AUT08450
END RCV$STATE$5;                                                         AUT08460
                                                                         AUT08470
RCV$STATE$6: PROCEDURE;                                                  AUT08480
                                                                         AUT08490
     /* LEVEL 3 PROCEDURE */                                             AUT08500
                                                                         AUT08510
     /* WHEN THE RECEIVER IS IN STATE 6 IT IS EXPECTING                 AUT08520
     TO GET AN 'SEL' CHARACTER.  IF THE SEL CHARACTER IS                 AUT08530
     CORRECT (BASED ON A TABLE LOOKUP), THE RECEIVER GOES                AUT08540
     TO STATE 3 (TO PROCESS TEXT); OTHERWISE, IT RETURNS                 AUT08550
     TO STATE 1 (WAITING FOR SOH). */                                    AUT08560
                                                                         AUT08570
     IF ((RCV$BYTE:=RCV$BYTE AND LOW$7$MASK) < 'A' OR                    AUT08580
          RCV$BYTE > 'D') AND RCV$BYTE <> 'H' AND                        AUT08590
          RCV$BYTE <> 'F' AND RCV$BYTE <> 'S' THEN                       AUT08600
          /* WE HAVE A BAD 'SEL' CHARACTER */                           AUT08610
          DO;                                                            AUT08620
               CHECK$RCV$TIMER=TRUE;                                     AUT08630
               RCV$SYNCH$TIMER=0;                                        AUT08640
```

```
                RCV$STATE=1;                                              AUT08650
            END;                                                          AUT08660
        ELSE  /* GOOD 'SEL' CHARACTER RECEIVED */                        AUT08670
            DO;                                                           AUT08680
                RCV$BP=(RCV$BP:=0) + RCV$BYTE; /* INITIALIZE BP */        AUT08690
                IF RCV$BYTE='A' OR RCV$BYTE='H' OR RCV$BYTE='S' THEN      AUT08700
                /* WE WANT TO OUTPUT ON THE TELETYPE */                   AUT08710
                    OUTPUT$DEVICE=TTY;                                    AUT08720
                ELSE                                                      AUT08730
                    IF RCV$BYTE='D' OR RCV$BYTE='F' THEN                  AUT08740
                    /* THEN IT'S A CARD MESSAGE */                        AUT08750
                        OUTPUT$DEVICE=CARD$PUNCH;                         AUT08760
                    ELSE /* IT MUST BE A MAG TAPE MESSAGE */              AUT08770
                        OUTPUT$DEVICE=MAG$TAPE;                           AUT08780
                RCV$TXT$CTR=0; /* INITIALIZE FOR NEW LINE BLOCK */        AUT08790
                 RCV$MID$MSG=TRUE;                                        AUT08800
                RCV$ERROR=FALSE;                                          AUT08810
                RCV$STATE=3;                                              AUT08820
                                                                          AUT08830
                /* CHECK FOR FLASH MESSAGE - RING BELL                    AUT08840
                FOR OPERATOR IF FLASH */                                  AUT08850
                IF RCV$BYTE='F' OR RCV$BYTE='S' THEN                      AUT08860
                    CALL ALARM(9);                                        AUT08870
            END;                                                          AUT08880
END RCV$STATE$6;                                                          AUT08890
                                                                          AUT08900
RCV$STATE$7: PROCEDURE;                                                   AUT08910
                                                                          AUT08920
    /* LEVEL 3 PROCEDURE */                                               AUT08930
                                                                          AUT08940
    /* WHEN THE RECEIVER IS IN STATE 7 IT IS EXPECTING A                  AUT08950
        'DEL' CHARACTER.  IF DEL RECEIVED THEN GOES INTO                  AUT08960
        STATE 3 (PROCESSING TEXT) OTHERWISE, IT GOES INTO                 AUT08970
        STATE 2 (WAITING FOR STX CHARACTER).  */                          AUT08980
                                                                          AUT08990
    IF RCV$BYTE=DEL THEN /* GO TO TEXT PROCESSING STATE */                AUT09000
        DO;                                                               AUT09010
            RCV$ERROR=FALSE;    /* INITIALIZE FOR NEW BLOCK */            AUT09020
            RCV$TXT$CTR, RCV$BP=0;                                        AUT09030
            RCV$BP=RCV$BP + RCV$BYTE;                                     AUT09040
            RCV$STATE=3;                                                  AUT09050
        END;                                                              AUT09060
    ELSE /* BYTE IS NOT A DEL - GO BACK TO STATE 2 */                     AUT09070
        DO;                                                               AUT09080
            RCV$SYNCH$TIMER=0;                                            AUT09090
            CHECK$RCV$TIMER=TRUE;                                         AUT09100
            RCV$STATE=2;                                                  AUT09110
        END;                                                              AUT09120
```

```
END RCV$STATE$7;                                                        AUT09130
                                                                        AUT09140
RCV$STATE$8: PROCEDURE;                                                 AUT09150
                                                                        AUT09160
     /* LEVEL 3 PROCEDURE */                                            AUT09170
                                                                        AUT09180
     /* IN STATE 8 THE RECEIVER IS LOOKING FOR VALID BLOCK             AUT09190
          PARITY.  IF BP IS VALID, RECEIVER FINISHES THE               AUT09200
          BLOCK AND ACKS FOR IT; OTHERWISE IT SENDS A                  AUT09210
          NAK (NEGATIVE ACKNOWLEDGEMENT          */                    AUT09220
                                                                        AUT09230
     IF RCV$BP <> RCV$BYTE THEN    /* BAD PARITY */                     AUT09240
          DO;                                                           AUT09250
               CALL SEND$CC(NAK);                                       AUT09260
               NAK$CTR=NAK$CTR + 1;                                     AUT09270
               IF NAK$CTR > 3 THEN CALL ALARM(1);                       AUT09280
               RCV$STATE=BLOCK$START$STATE;                             AUT09290
               RCV$SYNCH$TIMER=0;                                       AUT09300
               CHECK$RCV$TIMER=TRUE;                                    AUT09310
               RETURN;                                                  AUT09320
          END;                                                          AUT09330
      IF W$BUFFER$PTR > 1 THEN STILL$WRITING=TRUE;                      AUT09340
     IF STILL$WRITING AND RCV$MID$MSG THEN /* WRITER PROCESS IS NOT     AUT09350
                                        FINISHED WITH LAST BLOCK */     AUT09360
          DO;                                                           AUT09370
               CALL SEND$CC(WBT);                                       AUT09380
               NAK$CTR=0;                                               AUT09390
          END;                                                          AUT09400
     ELSE                                                               AUT09410
          DO;                                                           AUT09420
               IF XMT$ACK$1$2=1 THEN                                    AUT09430
                    DO;                                                 AUT09440
                         CALL SEND$CC(ACK1);                            AUT09450
                         NAK$CTR=0;                                     AUT09460
                         XMT$ACK$1$2=2;                                 AUT09470
                    END;                                                AUT09480
               ELSE                                                     AUT09490
                    DO;                                                 AUT09500
                         CALL SEND$CC(ACK2);                            AUT09510
                         NAK$CTR=0;                                     AUT09520
                         XMT$ACK$1$2=1;                                 AUT09530
                    END;                                                AUT09540
          END;                                                          AUT09550
     RCV$SYNCH$TIMER=0;                                                 AUT09560
     CHECK$RCV$TIMER=TRUE;                                              AUT09570
     IF RCV$MID$MSG THEN RCV$STATE=2;   /* SET NEXT RCV STATE */        AUT09580
     ELSE RCV$STATE=1;                                                  AUT09590
     WRITER$PROCESS(SCHEDULED)=TRUE;                                    AUT09600
```

```
      /* THIS SCHEDULES THE WRITER PROCESS TO DUMP THE BUFFER */      AUT09610
      RCV$BUFFER=RCV$TXT$CTR;  /* INSERT BUFFER LENGTH IN SLOT 1 */   AUT09620
      /* NOW SWITCH BUFFERS FOR THE RECEIVER LOGIC PROCESS */         AUT09630
      IF RCV$BUFFER$ADDR=.RW$BUFFER$1 THEN                            AUT09640
           RCV$BUFFER$ADDR=.RW$BUFFER$2;                              AUT09650
      ELSE RCV$BUFFER$ADDR=.RW$BUFFER$1;                              AUT09660
                                                                      AUT09670
END RCV$STATE$8;                                                      AUT09680
                                                                      AUT09690
RCV$STATE$9: PROCEDURE;                                               AUT09700
                                                                      AUT09710
      /* LEVEL 3 PROCEDURE */                                         AUT09720
                                                                      AUT09730
      /* IN STATE 9 THE RECEIVER IS EXPECTING THE SECOND OF A        AUT09740
      TWO-CHARACTER CONTROL SEQUENCE. IF THIS IS RECEIVED, THE        AUT09750
      CONTROL CHARACTERS ARE ACTED UPON, OTHERWISE AN ERROR           AUT09760
      CONDITION IS FLAGGED (RCV$ERROR).          */                   AUT09770
                                                                      AUT09780
      RCV$STATE=PREV$RCV$STATE;      /* NORMAL RETURN IS TO          AUT09790
                                        PREVIOUS STATE */             AUT09800
      IF PREV$RCV$BYTE <> RCV$BYTE THEN /* NOT A VALID CONTROL SEQ */ AUT09810
        DO;                                                           AUT09820
          RCV$ERROR=TRUE;                                             AUT09830
          RETURN;                                                     AUT09840
        END;                                                          AUT09850
      IF RCV$BYTE=ACK1 OR RCV$BYTE=ACK2 THEN                          AUT09860
      /* WE HAVE A VALID CONTROL CHAR SEQUENCE */                     AUT09870
        DO;                                                           AUT09880
          IF (RCV$BYTE=ACK1 AND RCV$ACK$1$2=1) OR                     AUT09890
             (RCV$BYTE=ACK2 AND RCV$ACK$1$2=2) THEN                   AUT09900
            DO;                                                       AUT09910
              AWAITING$ACK,XMT$WAIT=FALSE;                            AUT09920
              WBT$CTR,NAK$CTR,XMT$REP$CTR=0;                          AUT09930
              CHECK$ANS$TIMER=FALSE;                                  AUT09940
              XMT$ANS$TIMER=0;                                        AUT09950
              /* RESET TOGGLE */                                      AUT09960
              IF RCV$ACK$1$2=1 THEN RCV$ACK$1$2=2;                    AUT09970
              ELSE RCV$ACK$1$2=1;                                     AUT09980
              T$BUFFER$PTR=1;  /* RESET POINTER FOR REXMT */          AUT09990
              TRANSMIT$BUFFER=0;  /* MARKS BUFFER AS ACK'D */         AUT10000
              /* NOW SWITCH BUFFERS FOR TRANSMITTER PROCESS */        AUT10010
              IF TRANSMIT$BUFFER$ADDR=.T$BUFFER$1 THEN                AUT10020
                TRANSMIT$BUFFER$ADDR=.T$BUFFER$2;                     AUT10030
              ELSE TRANSMIT$BUFFER$ADDR=.T$BUFFER$1;                  AUT10040
            END;                                                      AUT10050
          ELSE                                                        AUT10060
            CALL SEND$CC(INV); /* ACK1/2 SEQUENCE WRONG */           AUT10070
        END;                                                          AUT10080
```

```
ELSE                                                                    AUT10090
  IF RCV$BYTE=NAK THEN                                                  AUT10100
    DO;                                                                 AUT10110
      IF NAK$CTR > 3 THEN CALL ALARM(1);                                AUT10120
      ELSE                                                              AUT10130
        DO;                                                             AUT10140
          XMT$ANS$TIMER=0;                                              AUT10150
          NAK$CTR=NAK$CTR + 1;                                          AUT10160
          T$BUFFER$PTR=1;  /* RESETS FOR RETRANSMISSION */              AUT10170
        END;                                                            AUT10180
    END;                                                                AUT10190
  ELSE                                                                  AUT10200
    IF RCV$BYTE=WBT THEN                                                AUT10210
      DO;                                                               AUT10220
        IF WBT$CTR > 3 THEN CALL ALARM(2);                              AUT10230
        ELSE                                                            AUT10240
          DO;                                                           AUT10250
            WBT$CTR=WBT$CTR + 1;                                        AUT10260
            XMT$ANS$TIMER=0;                                            AUT10270
          END;                                                          AUT10280
      END;                                                              AUT10290
    ELSE                                                                AUT10300
      IF RCV$BYTE=CAN THEN                                              AUT10310
        DO;                                                             AUT10320
          CALL ALARM(3);                                                AUT10330
          RCV$STATE=1;                                                  AUT10340
          XMT$ACK$1$2=1;                                                AUT10350
          CHECK$RCV$TIMER=TRUE;                                         AUT10360
          RCV$SYNCH$TIMER=0;                                            AUT10370
        END;                                                            AUT10380
      ELSE                                                              AUT10390
        IF RCV$BYTE=REP THEN                                            AUT10400
          DO;                                                           AUT10410
            IF RCV$MID$MSG OR STILL$WRITING THEN                        AUT10420
              DO;                                                       AUT10430
                IF STILL$WRITING THEN CALL SEND$CC(WBT);                AUT10440
                ELSE                                                    AUT10450
                  IF NAK$CTR > 0 THEN CALL SEND$CC(NAK);                AUT10460
                  ELSE                                                  AUT10470
                    DO;                                                 AUT10480
                      IF XMT$ACK$1$2=1 THEN                             AUT10490
                        CALL SEND$CC(ACK2);                             AUT10500
                      ELSE CALL SEND$CC(ACK1);                          AUT10510
                    END;                                                AUT10520
              END;                                                      AUT10530
            ELSE                                                        AUT10540
              CALL ALARM(4);                                            AUT10550
          END;                                                          AUT10560
```

```
                ELSE                                                    AUT10570
                  IF RCV$BYTE=RM THEN CALL ALARM(5);                    AUT10580
                  ELSE                                                  AUT10590
                    CALL ALARM(6);  /* INV WAS RECEIVED */              AUT10600
                                                                        AUT10610
END RCV$STATE$9;                                                        AUT10620
                                                                        AUT10630
/*   ***** RECEIVER LOGIC BEGINS EXECUTION HERE   ***** */             AUT10640
                                                                        AUT10650
RCV$BYTE=GET$RCVD$BYTE;                                                 AUT10660
IF RCV$STATE < 5 THEN                                                   AUT10670
    DO; /* FIRST DETERMINE ACTION BASED ON RCVD BYTE AND               AUT10680
           RCV STATE */                                                 AUT10690
        RCV$BYTE=RCV$BYTE AND 0FFH; /* PERFORM AND TO INSURE           AUT10700
                                       THAT PARITY IS SET */            AUT10710
        IF PARITY EVEN THEN                                             AUT10720
            DO;                                                         AUT10730
                IF(LOOKUP:=(RCV$BYTE AND LOW$7$MASK))>7 THEN            AUT10740
                    IF(LOOKUP:=LOOKUP-9) > 21 THEN                      AUT10750
                        LOOKUP=20;                                      AUT10760
                /* THIS HAS MAPPED THE CONTROL CHARACTERS INTO          AUT10770
                    THE INTEGERS 1-21 */                                AUT10780
            END;                                                        AUT10790
        ELSE LOOKUP=22;  /* BYTE IS NOT CONTROL CHARACTER */            AUT10800
        /* PERFORM TABLE LOOKUP BASED ON RCV STATE AND CHARACTER */     AUT10810
        DO CASE RCV$STATE;                                              AUT10820
            ; /* CASE 0 NOT USED */                                     AUT10830
            DO; /* CASE 1 - CHECK FOR LOSS OF SYNCH. IF SYNCH          AUT10840
                   OK THEN DO LOOKUP FOR RCV STATE 1 */                 AUT10850
                IF CHECK$RCV$TIMER AND RCV$SYNCH$TIMER EXPIRED THEN     AUT10860
                    DO;                                                 AUT10870
                        RCV$STATE=5;                                    AUT10880
                        RECEIVE$LOGIC$PROCESS(SCHEDULED)=TRUE;          AUT10890
                        RETURN;                                         AUT10900
                    END;                                                AUT10910
                ACTION=TABLE$1(LOOKUP);                                 AUT10920
            END;                                                        AUT10930
                                                                        AUT10940
            DO; /* CASE 2 - CHECK FOR SYNC LOSS.  IF SYNCH             AUT10950
                   OK THEN DO LOOKUP FOR RCV STATE 2 */                 AUT10960
                IF CHECK$RCV$TIMER AND RCV$SYNCH$TIMER EXPIRED THEN     AUT10970
                    DO;                                                 AUT10980
                        RCV$STATE=5;                                    AUT10990
                        RECEIVE$LOGIC$PROCESS(SCHEDULED)=TRUE;          AUT11000
                        RETURN;                                         AUT11010
                    END;                                                AUT11020
                ACTION=TABLE$2(LOOKUP);                                 AUT11030
            END;                                                        AUT11040
```

```
                ACTION=TABLE$3(LOOKUP);  /* CASE 3 - RCV STATE 3 */
                ACTION=TABLE$4(LOOKUP); /* CASE 4 - RCV STATE 4  */
            END; /* OF CASE RCVSTATE */
            CALL PERFORM$ACTION;
        END;

    ELSE        /* RECEIVER STATE IS BETWEEN 5 AND 9 */
        DO CASE (ACTION:=RCV$STATE - 5);

            /* CASE 0 - RCV$STATE =5 - ATTEMPTING TO
                        ACHIEVE SYNCHRONIZATION */
            CALL RCV$STATE$5;

            /* CASE 1, RCV$STATE = 6 - EXPECTING 'SEL' BYTE */
            CALL RCV$STATE$6;

            /* CASE 2, RCV$STATE=7 - EXPECTING 'DEL' BYTE */
            CALL RCV$STATE$7;

            /* CASE 3, RCV$STATE=8 - EXPECTING ELOCK PARITY BYTE */
            CALL RCV$STATE$8;

            /* CASE 4, RCV$STATE=9 - PROCESSING THE SECOND CHAR
            IN A TWO-CHARACTER CONTROL SEQUENCE */
            CALL RCV$STATE$9;

        END; /* OF CASE ACTION */

END RECEIVE$LOGIC;

WRITER: PROCEDURE;

    /*   LEVEL 2 PROCEDURE   */

    /*   WRITES BYTES (ONE AT A TIME) ON THE SELECTED
         OUTPUT DEVICE AND RESCHEDULES ITSELF WHENEVER
         MORE BYTES ARE TO BE DUMPED FROM A WRITE BUFFER. */

    CALL WRITE$BYTE(WRITE$BUFFER(W$BUFFER$PTR));
    W$BUFFER$PTR=W$BUFFER$PTR + 1;
    IF W$BUFFER$PTR > WRITE$BUFFER THEN /* DONE WRITING CURRENT BUFF */
        DO;
            WRITE$BUFFER=0;
            /* SET LENGTH OF BUFFER- IT'S ALREADY WRITTEN.
                THEN FLIP BUFFERS     */
            IF WRITE$BUFFER$ADDR=.RW$BUFFER$1 THEN
                WRITE$BUFFER$ADDR=.RW$BUFFER$2;
```

```
                ELSE WRITE$BUFFER$ADDR=.RW$BUFFER$1;                    AUT11530
                 W$BUFFER$PTR=1;                                        AUT11540
                 STILL$WRITING=FALSE;                                   AUT11550
                IF WRITE$BUFFER=0 THEN /* GO TO SLEEP - NO              AUT11560
                                          RESCHEDULE NEEDED */          AUT11570
                  DO;                                                   AUT11580
                     IF NOT RCV$MID$MSG THEN OUTPUT$DEVICE=NOT$SELECTED;AUT11590
                     WRITER$PROCESS(SCHEDULED)=FALSE;                   AUT11600
                  END;                                                  AUT11610
          END;                                                          AUT11620
                                                                        AUT11630
END WRITER;                                                             AUT11640
                                                                        AUT11650
OPERATOR$INPUT: PROCEDURE;                                              AUT11660
                                                                        AUT11670
    /*   LEVEL 2 PROCEDURE   */                                         AUT11680
                                                                        AUT11690
    /* INTERPRETS COMMANDS FROM THE OPERATOR CONSOLE -                 AUT11700
         T= TRANSMIT NEW MESSAGE.                                       AUT11710
         C= CANCEL CURRENTLY TRANSMITTED MSG.                           AUT11720
         B= REBOOT (RE-INITIALIZE).      */                            AUT11730
                                                                        AUT11740
    DECLARE CHARACTER BYTE;                                             AUT11750
                                                                        AUT11760
    RETURN;  /* INSERTED FOR TEST/DEBUG ONLY */                         AUT11770
    IF(CHARACTER:=CRT$IN)='T' THEN  /* TRANSMIT NEW MSG */              AUT11780
         SENDING=TRUE;                                                  AUT11790
    ELSE                                                                AUT11800
         IF CHARACTER='C' THEN  /* CANCEL THIS MSG */                  AUT11810
              DO;                                                       AUT11820
                   CALL SEND$CC(CAN);                                   AUT11830
                   CALL INITIALIZE$XMTR;                                AUT11840
              END;                                                      AUT11850
         ELSE                                                           AUT11860
              IF CHARACTER='B' THEN  /* REBOOT THE SYSTEM */           AUT11870
                   GO TO RESTART;                                       AUT11880
              /* ELSE NOTHING - ONLY T,C, AND B VALID COMMANDS */       AUT11890
                                                                        AUT11900
END OPERATOR$INPUT;                                                     AUT11910
                                                                        AUT11920
OPERATOR$OUTPUT: PROCEDURE;                                             AUT11930
                                                                        AUT11940
    /*   LEVEL 2 PROCEDURE   */                                         AUT11950
                                                                        AUT11960
    /* OUTPUTS MESSAGES TO THE CRT ONE BYTE AT A TIME:                 AUT11970
                                                                        AUT11980
         STATE 1: SEND BELL                                             AUT11990
         STATE 2: SEND ONE LETTER DIAGNOSTIC MSG.                       AUT12000
```

```
          STATE 3: SEND CR.                                                  AUT12010
          STATE 4: SEND LF                              */                   AUT12020
                                                                             AUT12030
     DO CASE OUTPUT$STATE;                                                   AUT12040
                                                                             AUT12050
          ;    /* CASE 0 NOT USED */                                        AUT12060
                                                                             AUT12070
          DO;  /* CASE 1 OUTPUT BELL ALARM */                               AUT12080
               CALL CRT$OUT(BELL);                                           AUT12090
               OUTPUT$STATE=2;                                               AUT12100
          END;                                                               AUT12110
                                                                             AUT12120
          DO;  /* CASE 2 - OUTPUT DIAGNOSTIC CHARACTER */                   AUT12130
               CALL CRT$OUT(ALARM$MSG);                                      AUT12140
               OUTPUT$STATE=3;                                               AUT12150
          END;                                                               AUT12160
                                                                             AUT12170
          DO;  /* CASE 3 - OUTPUT <CR> */                                   AUT12180
               CALL CRT$OUT(CR);                                             AUT12190
               OUTPUT$STATE=4;                                               AUT12200
          END;                                                               AUT12210
                                                                             AUT12220
          DO;  /* CASE 4 - OUTPUT <LF> AND GO TC SLEEP */                   AUT12230
               CALL CRT$OUT(LF);                                             AUT12240
               OPERATOR$OUTPUT$PROCESS(SCHEDULED)=FALSE;                     AUT12250
          END;                                                               AUT12260
                                                                             AUT12270
     END; /* OF CASE OUTPUT STATE */                                         AUT12280
                                                                             AUT12290
END OPERATOR$OUTPUT;                                                         AUT12300
                                                                             AUT12310
TRANSMIT$LOGIC: PROCEDURE;                                                   AUT12320
                                                                             AUT12330
     /*   LEVEL 2 PROCEDURE   */                                             AUT12340
                                                                             AUT12350
     /*  BUILDS BLOCKS FOR OUTGOING MESSAGE TRANSMISSION (ONE               AUT12360
         BYTE AT A TIME).  INSERTS NEEDED FRAMING BYTES (SOH,STX,           AUT12370
         SEL, DEL, EM, ETB, ETX, AND BP), CALCULATES ODD PARITY             AUT12380
         FOR TEXT BYTES, SCHEDULES TRANSMITTER PROCESS WHEN LINE            AUT12390
         BLOCKS BECOME AVAILABLE FOR TRANSMISSION, AND SCHEDULES            AUT12400
         THE OPERATOR OUTPUT PROCESS WHEN ALARM CONDITIONS OCCUR.           AUT12410
         THE TRANSMIT LOGIC PROCESS IS A 5 STATE AUTOMATON WITH             AUT12420
         THE FOLLOWING STATES:                                              AUT12430
               1. PROCESSING BEGINNING OF A NEW MESSAGE.                    AUT12440
               2. WAITING FOR SECOND CHARACTER OF A MESSAGE.                AUT12450
               3. WAITING FOR THIRD CHARACTER OF A MESSAGE.                 AUT12460
               4. PROCESSING TEXT IN THE MIDDLE OF A MESSAGE.               AUT12470
               5. PROCESSING THE BEGINNING OF AN INTERMEDIATE LINE BLOCK.   AUT12480
```

```
        */                                                              AUT12490
                                                                        AUT12500
                                                                        AUT12510
EOM: PROCEDURE(CHARACTER) BYTE;                                         AUT12520
                                                                        AUT12530
     /* LEVEL 3 PROCEDURE */                                            AUT12540
                                                                        AUT12550
     /* CHECKS FOR 8 LINE FEEDS FOLLOWED BY 4 N'S IN A ROW.            AUT12560
     RETURNS TRUE ONLY IF THIS HAS OCCURED (THIS IS THE END OF A        AUT12570
     MESSAGE IN AUTODIN).  OTHERWISE, IT RETURNS FALSE.   */            AUT12580
                                                                        AUT12590
     DECLARE CHARACTER BYTE;                                            AUT12600
                                                                        AUT12610
     IF EOM$STATE=1 THEN                                                AUT12620
          DO;                                                           AUT12630
               IF CHARACTER=08AH THEN                                   AUT12640
                /* 08AH=ODD PARITY  LF */                               AUT12650
                    DO;                                                 AUT12660
                         LF$CTR=LF$CTR + 1;                             AUT12670
                         IF LF$CTR=8 THEN /* WE HAVE 8 LF'S IN A ROW*/  AUT12680
                              GO TO EOM$STATE 2 AND COUNT N'S */        AUT12690
                              DO;                                       AUT12700
                                   LF$CTR=0;                            AUT12710
                                   EOM$STATE=2;                         AUT12720
                              END;                                      AUT12730
                    END;                                                AUT12740
               ELSE LF$CTR=0;                                           AUT12750
          END;                                                          AUT12760
     ELSE                                                               AUT12770
          DO;                                                           AUT12780
               IF CHARACTER=0CEH THEN                                   AUT12790
                /* 0CEH=ODD PARITY 'N' */                               AUT12800
                    DO;                                                 AUT12810
                         N$CTR=N$CTR + 1;                               AUT12820
                         IF N$CTR =4 THEN /* WE'RE AT END OF MSG */    AUT12830
                              DO;                                       AUT12840
                                   N$CTR=0;                             AUT12850
                                   EOM$STATE=1;/* RESET NEXT TIME */   AUT12860
                                   RETURN TRUE;                         AUT12870
                              END;                                      AUT12880
                    END;                                                AUT12890
               ELSE  /* NOT AN EOM - GO BACK TO STATE 1  */            AUT12900
                    DO;                                                 AUT12910
                         N$CTR=0;                                       AUT12920
                         EOM$STATE=1;                                   AUT12930
                    END;                                                AUT12940
          END;                                                          AUT12950
     RETURN FALSE;                                                      AUT12960
```

```
END EOM;                                                                AUT12970
                                                                        AUT12980
XMT$STATE$1: PROCEDURE;                                                 AUT12990
                                                                        AUT13000
     /* LEVEL 3 PROCEDURE */                                            AUT13010
                                                                        AUT13020
     /*  STATE 1 IS WAITING TO START A NEW MSG FOR TRANSMISSION  */     AUT13030
                                                                        AUT13040
     IF (XMT$BYTE:=GET$BYTE$TO$XMT)=SPACE THEN /* GOBBLE SPACES       AUT13050
                                        AT THE BEGINNING OF TAPE */     AUT13060
          RETURN;                                                       AUT13070
     /* OTHERWISE GET A NEW BUFFER AND PUT AN 'SOH' IN SLOT 1 */        AUT13080
     IF XMT$LOGIC$BUFFER$ADDR=.T$BUFFER$1 THEN                          AUT13090
          XMT$LOGIC$BUFFER$ADDR=.T$BUFFER$2;                            AUT13100
     ELSE XMT$LOGIC$BUFFER$ADDR=.T$BUFFER$1;                            AUT13110
     XMT$LOGIC$BUFFER(1)=SOH;                                           AUT13120
     XMT$BYTE=XMT$BYTE AND 0FFH; /* ENSURE PARITY SET */                AUT13130
     IF PARITY EVEN THEN                                                AUT13140
          XMT$BYTE=XMT$BYTE OR PARITY$MASK; /* INSERT ODD PARITY */     AUT13150
     XMT$LOGIC$BUFFER(3)=XMT$BYTE;                                      AUT13160
     XMT$BP=(XMT$BP:=0) + XMT$BYTE; /* INITIALIZE AND COMPUTE BP */     AUT13170
     XMT$STATE=2; /* SET NEXT STATE */                                  AUT13180
                                                                        AUT13190
END XMT$STATE$1;                                                        AUT13200
                                                                        AUT13210
XMT$STATE$2: PROCEDURE;                                                 AUT13220
                                                                        AUT13230
     /* LEVEL 3 PROCEDURE */                                            AUT13240
                                                                        AUT13250
     /* PROCESSES THE SECOND BYTE OF A TRANSMITTED MESSAGE */           AUT13260
                                                                        AUT13270
     IF CAN$FLAG THEN /* CANCEL OUTGOING MESSAGE */                     AUT13280
          DO;                                                           AUT13290
               CAN$FLAG=FALSE;                                          AUT13300
               XMT$STATE=1;                                             AUT13310
               CALL SEND$CC(CAN);                                       AUT13320
               SENDING=FALSE;                                           AUT13330
          END;                                                          AUT13340
     ELSE /* PROCESS SECOND BYTE OF MESSAGE */                         AUT13350
          DO;                                                           AUT13360
               XMT$BYTE=((XMT$BYTE:=GET$BYTE$TO$XMT) AND 0FFH);         AUT13370
               IF PARITY EVEN THEN                                      AUT13380
                    XMT$BYTE=XMT$BYTE OR PARITY$MASK;                   AUT13390
               XMT$LOGIC$BUFFER(4)=XMT$BYTE;                            AUT13400
               XMT$BP=XMT$BP + XMT$BYTE;                                AUT13410
               XMT$STATE=3; /* SET NEXT STATE */                        AUT13420
          END;                                                          AUT13430
                                                                        AUT13440
```

```
END XMT$STATE$2;                                                        AUT13450
                                                                        AUT13460
XMT$STATE$3: PROCEDURE;                                                 AUT13470
                                                                        AUT13480
     /* LEVEL 3 PROCEDURE */                                            AUT13490
                                                                        AUT13500
     /* CHECKS THE THIRD CHARACTER OF THE OUTGOING MESSAGE TO           AUT13510
     SEE IF IT IS A VALID LMF CHARACTER.  IF IT IS, IT IS               AUT13520
     CONVERTED TO THE CORRESPONDING 'SEL' CHARACTER.  THE LMF           AUT13530
     CHARACTER IS INSERTED IN THE THIRD TEXT SLOT OF THE                AUT13540
     LINE BLOCK; THE SEL CHARACTER IS INSERTED IN THE SECOND            AUT13550
     FRAMING SLOT OF THE LINE BLOCK.    */                             AUT13560
                                                                        AUT13570
     DECLARE SEL BYTE,          /* USED TO LOOK UP THE SEL CHAR */     AUT13580
          CHECK BYTE;           /* USED TO CHECK CORRECTNESS OF SEL */ AUT13590
                                                                        AUT13600
     CHECK$LMF: PROCEDURE BYTE;                                         AUT13610
                                                                        AUT13620
          /* LEVEL 4 PROCEDURE */                                       AUT13630
                                                                        AUT13640
          /* RETURNS TRUE AND SETS THE SEL CORRECTLY IF THE LMF        AUT13650
          CHARACTER (THIRD BYTE OF THE OUTGOING MSG) WAS CORRECT.       AUT13660
          RETURNS FALSE OTHERWISE.    */                              AUT13670
                                                                        AUT13680
          IF XMT$BYTE > 40H AND XMT$BYTE < 4AH AND XMT$BYTE <> 'E'      AUT13690
               AND XMT$BYTE <> 'H' THEN                                 AUT13700
               DO;                                                      AUT13710
                    SEL=SEL$LOOKUP$1(XMT$BYTE - 41H);                   AUT13720
                    RETURN TRUE;                                        AUT13730
               END;                                                     AUT13740
          ELSE                                                          AUT13750
               IF XMT$BYTE > 50H AND XMT$BYTE < 55H THEN                AUT13760
                    DO;                                                 AUT13770
                         SEL=SEL$LOOKUP$2(XMT$BYTE - 51H);              AUT13780
                         RETURN TRUE;                                   AUT13790
                    END;                                                AUT13800
               ELSE /* BAD LMF CHARACTER - NO CORRESPONDING SEL */      AUT13810
                    RETURN FALSE;                                       AUT13820
     END CHECK$LMF;                                                     AUT13830
                                                                        AUT13840
     IF CAN$FLAG THEN /* CANCEL MESSAGE */                             AUT13850
          DO;                                                           AUT13860
               CAN$FLAG=FALSE;                                          AUT13870
               XMT$STATE=1;                                             AUT13880
               CALL SEND$CC(CAN);                                       AUT13890
               SENDING=FALSE;                                           AUT13900
          END;                                                          AUT13910
     ELSE                                                               AUT13920
```

```
          DO;                                                              AUT13930
               XMT$BYTE=GET$BYTE$TO$XMT;                                   AUT13940
               IF (CHECK:=CHECK$LMF) THEN /* GOOD LMF CHARACTER */         AUT13950
                    DO;                                                    AUT13960
                         SEL=SEL AND OFFH; /* ENSURE PARITY SET */       AUT13970
                         IF NOT (PARITY EVEN) THEN                         AUT13980
                              SEL=SEL OR PARITY$MASK;                      AUT13990
                         XMT$BP=XMT$BP + SEL;                              AUT14000
                         XMT$LOGIC$BUFFER(2)=SEL;                          AUT14010
                         XMT$BYTE=XMT$BYTE AND OFFH;                       AUT14020
                         IF PARITY EVEN THEN                               AUT14030
                              XMT$BYTE=XMT$BYTE OR PARITY$MASK;            AUT14040
                         XMT$BP=XMT$BP + XMT$BYTE;                         AUT14050
                         XMT$LOGIC$BUFFER(5)=XMT$BYTE;                     AUT14060
                         XMT$TXT$CTR=6;                                    AUT14070
                         XMT$STATE=4;                                      AUT14080
                    END;                                                   AUT14090
               ELSE CALL ALARM(8);                                         AUT14100
          END;                                                             AUT14110
                                                                           AUT14120
END XMT$STATE$3;                                                           AUT14130
                                                                           AUT14140
XMT$STATE$4: PROCEDURE;                                                    AUT14150
                                                                           AUT14160
     /* LEVEL 3 PROCEDURE */                                               AUT14170
                                                                           AUT14180
     /* PROCESSES TEXT OF AN OUTGOING MESSAGE.  BUILDING THE              AUT14190
     INFORMATIONAL PORTION OF THE LINE BLOCKS.     */                      AUT14200
                                                                           AUT14210
     DECLARE CHECK BYTE;                                                   AUT14220
                                                                           AUT14230
     IF CAN$FLAG THEN /* CANCEL OUTGOING MESSAGE */                       AUT14240
          DO;                                                              AUT14250
               CAN$FLAG=FALSE;                                             AUT14260
               XMT$STATE=1;                                                AUT14270
               CALL SEND$CC(CAN);                                          AUT14280
               SENDING=FALSE;                                              AUT14290
          END;                                                             AUT14300
     ELSE                                                                  AUT14310
          DO;                                                              AUT14320
               XMT$BYTE=((XMT$BYTE:=GET$BYTE$TO$XMT) AND OFFH);            AUT14330
               IF PARITY EVEN THEN                                         AUT14340
                    XMT$BYTE=XMT$BYTE OR PARITY$MASK;                      AUT14350
               XMT$LOGIC$BUFFER(XMT$TXT$CTR)=XMT$BYTE;                     AUT14360
               XMT$BP=XMT$BP + XMT$BYTE;                                   AUT14370
               XMT$TXT$CTR=XMT$TXT$CTR + 1;                                AUT14380
               IF (XMT$TXT$CTR > 82) OR (CHECK:=EOM(XMT$BYTE)) THEN        AUT14390
                    DO;                                                    AUT14400
```

```
                        IF CHECK THEN /* WE'RE AT THE END OF MSG */AUT14410
                          DO;                                     AUT14420
                             IF XMT$TXT$CTR < 82 THEN             AUT14430
                                  DO;                             AUT14440
                                       XMT$LOGIC$BUFFER(          AUT14450
                                            XMT$TXT$CTR)=EM;      AUT14460
                                       XMT$BP=XMT$BP + EM;        AUT14470
                                      SENDING=FALSE;              AUT14480
                                      XMT$TXT$CTR=XMT$TXT$CTR + 1;AUT14490
                                      INPUT$DEVICE=NOT$SELECTED;  AUT14500
                                     IF VIRTUAL THEN SENDING=TRUE;AUT14510
                                  END;                            AUT14520
                             XMT$STATE=1;/* SET STATE FOR NEW MSG*/AUT14530
                             XMT$LOGIC$BUFFER(XMT$TXT$CTR)=ETX;   AUT14540
                             XMT$BP=XMT$BP + ETX;                 AUT14550
                             XMT$TXT$CTR=XMT$TXT$CTR + 1;         AUT14560
                          END;                                    AUT14570
                        ELSE                                      AUT14580
                             DO;                                  AUT14590
                                  XMT$STATE=5;                    AUT14600
                                 XMT$LOGIC$BUFFER(XMT$TXT$CTR)=ETB;AUT14610
                                  XMT$BP=XMT$BP + ETB;            AUT14620
                                  XMT$TXT$CTR=XMT$TXT$CTR + 1;    AUT14630
                             END;                                 AUT14640
                        XMT$ANS$TIMER=0;                          AUT14650
                        CHECK$ANS$TIMER=TRUE;                     AUT14660
                        XMT$LOGIC$BUFFER(XMT$TXT$CTR)=XMT$BP;     AUT14670
                        XMT$LOGIC$BUFFER=XMT$TXT$CTR; /* INSERTS  AUT14680
                             LENGTH AT BEGINNING OF BUFFER */     AUT14690
                        IF AWAITING$ACK THEN                      AUT14700
                             XMT$WAIT=TRUE;                       AUT14710
                   END;                                           AUT14720
              END;                                                AUT14730
END XMT$STATE$4;                                                  AUT14740
                                                                  AUT14750
XMT$STATE$5: PROCEDURE;                                           AUT14760
                                                                  AUT14770
     /* LEVEL 3 PROCEDURE */                                      AUT14780
                                                                  AUT14790
     /* PROCESSING THE FRAMING CHARACTERS REQUIRED AT THE        AUT14800
     BEGINNING OF INTERMEDIATE LINE BLOCKS OF OUTGOING MSG */     AUT14810
                                                                  AUT14820
     DECLARE CHECK BYTE ; /* USED TO CHECK FOR END OF MSG */      AUT14830
     /* GET NEW BUFFER */                                         AUT14840
     IF XMT$LOGIC$BUFFER$ADDR=.T$BUFFER$1 THEN                    AUT14850
          XMT$LOGIC$BUFFER$ADDR=.T$BUFFER$2;                      AUT14860
     ELSE XMT$LOGIC$BUFFER$ADDR=.T$BUFFER$1;                      AUT14870
     XMT$LOGIC$BUFFER(1)=STX; /* INSERT 'STX' */                  AUT14880
```

```
      XMT$LOGIC$BUFFER(2)=DEL;                                          AUT14890
      XMT$BYTE=((XMT$BYTE:=GET$BYTE$TO$XMT) AND 0FFH);                  AUT14900
      IF PARITY EVEN THEN                                               AUT14910
           XMT$BYTE=XMT$BYTE OR PARITY$MASK; /* INSERT ODD PARITY */    AUT14920
      XMT$LOGIC$BUFFER(3)=XMT$BYTE;                                     AUT14930
      XMT$BP=(XMT$BP:=0) + DEL + XMT$BYTE;                              AUT14940
       XMT$TXT$CTR=4;                                                   AUT14950
      IF (CHECK:=EOM(XMT$BYTE)) THEN /* WE'RE AT END OF A MSG */        AUT14960
                                                                        AUT14970
           DO;                                                          AUT14980
                XMT$LOGIC$BUFFER(4)=EM;                                 AUT14990
                 SENDING=FALSE;                                         AUT15000
                XMT$LOGIC$BUFFER(5)=ETX;                                AUT15010
                XMT$BP=XMT$BP + EM + ETX;                               AUT15020
                XMT$LOGIC$BUFFER(6)=XMT$BP;                             AUT15030
                XMT$LOGIC$BUFFER=6; /* STORE LENGTH OF BUFFER */        AUT15040
                XMT$ANS$TIMER=0;                                        AUT15050
                CHECK$ANS$TIMER=TRUE;                                   AUT15060
                IF AWAITING$ACK THEN XMT$WAIT=TRUE;                     AUT15070
                     /* TRANSMIT LOGIC PROCESS HAS 2 BLOCKS PENDING    AUT15080
                     IT MUST WAIT UNTIL 1 OF THEM IS ACK'D.  */         AUT15090
                XMT$STATE=1;                                            AUT15100
           END;                                                         AUT15110
      ELSE XMT$STATE=4; /* CONTINUE PROCESSING TEXT */                  AUT15120
END XMT$STATE$5;                                                        AUT15130
                                                                        AUT15140
/* ***** TRANSMIT LOGIC PROCESS BEGINS EXECUTION HERE ***** */         AUT15150
                                                                        AUT15160
IF AWAITING$ACK AND XMT$ANS$TIMER EXPIRED AND                           AUT15170
     CHECK$ANS$TIMER THEN                                               AUT15180
     DO;                                                                AUT15190
           CALL SEND$CC(REP);                                           AUT15200
           XMT$ANS$TIMER=0;                                             AUT15210
           XMT$REP$CTR=XMT$REP$CTR + 1;                                 AUT15220
           IF XMT$REP$CTR > 3 THEN CALL ALARM(7);                       AUT15230
           RETURN;                                                      AUT15240
     END;                                                               AUT15250
IF XMT$WAIT THEN RETURN; /* MUST SLEEP UNTIL A BLOCK IS ACK'D */        AUT15260
                                                                        AUT15270
DO CASE XMT$STATE;                                                      AUT15280
                                                                        AUT15290
     ;    /* CASE 0 NOT USED */                                        AUT15300
                                                                        AUT15310
     CALL XMT$STATE$1;   /* CASE 1*/                                    AUT15320
                                                                        AUT15330
     CALL XMT$STATE$2;   /* CASE 2 */                                   AUT15340
                                                                        AUT15350
     CALL XMT$STATE$3;   /* CASE 3 */                                   AUT15360
```

```
            CALL XMT$STATE$4;    /* CASE 4 */                          AUT15370
                                                                       AUT15380
            CALL XMT$STATE$5;    /* CASE 5 */                          AUT15390
                                                                       AUT15400
        END;        /* OF CASE XMT$STATE */                            AUT15410
                                                                       AUT15420
                                                                       AUT15430
    END TRANSMIT$LOGIC;                                                AUT15440
                                                                       AUT15450
    TRANSMITTER: PROCEDURE;                                            AUT15460
                                                                       AUT15470
        /*    LEVEL 2 PROCEDURE    */                                  AUT15480
                                                                       AUT15490
        /*   TRANSMITS BYTES ONE AT A TIME ACCORDING TO THE PRIORITY:   AUT15500
             1) CONTROL CHARACTER TO SEND, 2) LINE BLOCK DATA TO       AUT15510
             SEND, OR 3) SYN TO SEND.  IS AUTOMATICALLY SCHEDULED      AUT15520
             EVERY TIME THE USAT IS READY.       */                    AUT15530
                                                                       AUT15540
        IF (CC$PTR1 <> CC$PTR2) AND (T$BUFFER$PTR <> 2 OR T$BUFFER$PTR <>  AUT15550
            (T$BUFFER$LENGTH:=TRANSMIT$BUFFER)) THEN                   AUT15560
             /* THE ABOVE LINES CHECK TO SEE IF THERE ARE CONTROL CHARACTERS  AUT15570
             TO SEND, CHECKS TO SEE THAT WE'RE NOT FRAMING A BLOCK, AND    AUT15580
             ASSIGNS BUFFER LENGTH THE VALUE STORED IN THE FIRST ELEMENT   AUT15590
             OF THE TRANSMIT BUFFER (WHERE TRANSMIT LOGIC PROCESS STORED   AUT15600
             THE LENGTH OF THE BUFFER TO TRANSMIT.    */                  AUT15610
                                                                       AUT15620
             DO; /* SEND A CONTROL CHARACTER */                        AUT15630
             CALL SEND$BYTE(CONCHAR$BUFFER(CC$PTR1));                  AUT15640
             IF (CC$PTR1:=CC$PTR1 + 1) > 7 THEN                        AUT15650
                  CC$PTR1=0;     /* RESET THE DEQUE */                 AUT15660
             END;                                                      AUT15670
        ELSE                                                           AUT15680
             IF T$BUFFER$PTR <= TRANSMIT$BUFFER THEN                   AUT15690
                  /* SEND ONE BYTE OF THE LINE BLOCK */                AUT15700
                  DO;                                                  AUT15710
                       IF T$BUFFER$PTR=1 THEN AWAITING$ACK = TRUE;     AUT15720
                       CALL SEND$BYTE(TRANSMIT$BUFFER(T$BUFFER$PTR));  AUT15730
                       T$BUFFER$PTR=T$BUFFER$PTR + 1;                  AUT15740
                  END;                                                 AUT15750
             ELSE                                                      AUT15760
                  CALL SEND$BYTE(SYN);                                 AUT15770
                                                                       AUT15780
    END TRANSMITTER;                                                   AUT15790
                                                                       AUT15800
    POLL$DEVICES: PROCEDURE;                                           AUT15810
                                                                       AUT15820
        /*    LEVEL 2 PROCEDURE    */                                  AUT15830
                                                                       AUT15840
```

```
/*  PERFORMED ONCE EVERY LOOP OF THE EXECUTIVE PROCESS                  AUT15850
    SCHEDULER.  CHECKS TO SEE IF LOCAL I/O DEVICES ARE                  AUT15860
    READY AND MARKS THEM READY FOR THE SCHEDULER.  */                   AUT15870
                                                                        AUT15880
                                                                        AUT15890
IF OUTPUT$DEVICE <> NOT$SELECTED THEN                                   AUT15900
    DO;                                                                 AUT15910
        IF OUTPUT$DEVICE=TTY THEN                                       AUT15920
            DO;                                                         AUT15930
                IF NOT ROR(INPUT(1),2) THEN                             AUT15940
                    WRITER$PROCESS(DEVICE$READY)=TRUE;                  AUT15950
                ELSE WRITER$PROCESS(DEVICE$READY)=FALSE;                AUT15960
            END;                                                        AUT15970
        ELSE                                                            AUT15980
        IF OUTPUT$DEVICE=MAG$TAPE THEN                                  AUT15990
                DO;                                                     AUT16000
                IF NOT ROR(INPUT(3),2) THEN                             AUT16010
                        WRITER$PROCESS(DEVICE$READY)=TRUE;              AUT16020
                    ELSE                                                AUT16030
                        WRITER$PROCESS(DEVICE$READY)=FALSE;             AUT16040
                END;                                                    AUT16050
            ELSE  /* CHECK THE CARD PUNCH FOR STATUS */                 AUT16060
            DO;                                                         AUT16070
                    IF NOT ROR(INPUT(4),2) THEN                         AUT16080
                        WRITER$PROCESS(DEVICE$READY)=TRUE;              AUT16090
                    ELSE                                                AUT16100
                    WRITER$PROCESS(DEVICE$READY)=FALSE;                 AUT16110
                END;                                                    AUT16120
    END;                                                                AUT16130
                                                                        AUT16140
IF INPUT$DEVICE=NOT$SELECTED THEN                                       AUT16150
    DO;                                                                 AUT16160
        IF NOT(INPUT(TTY$STATUS)) THEN                                  AUT16170
            INPUT$DEVICE=TTY;                                           AUT16180
        ELSE                                                            AUT16190
            IF NOT(INPUT(MAG$TAPE)) THEN                                AUT16200
                INPUT$DEVICE=MAG$TAPE;                                  AUT16210
            ELSE                                                        AUT16220
                IF NOT(INPUT(CARD$PUNCH)) THEN                          AUT16230
                    INPUT$DEVICE=CARD$PUNCH;                            AUT16240
        IF INPUT$DEVICE=NOT$SELECTED THEN                               AUT16250
            TRANSMIT$LOGIC$PROCESS(DEVICE$READY)=FALSE;                 AUT16260
        ELSE                                                            AUT16270
            TRANSMIT$LOGIC$PROCESS(DEVICE$READY)=TRUE;                  AUT16280
    END;                                                                AUT16290
ELSE                                                                    AUT16300
    DO;                                                                 AUT16310
        IF INPUT$DEVICE=TTY THEN                                        AUT16320
```

```
                    DO;                                                    AUT16330
                         IF NOT INPUT(1) THEN                              AUT16340
                              WRITER$PROCESS(DEVICE$READY)=TRUE;           AUT16350
                         ELSE                                              AUT16360
                         WRITER$PROCESS(DEVICE$READY)=FALSE;               AUT16370
                    END;                                                   AUT16380
               ELSE                                                        AUT16390
               IF INPUT$DEVICE=MAG$TAPE THEN                               AUT16400
                    DO;                                                    AUT16410
                         IF NOT INPUT(3) THEN                              AUT16420
                                   WRITER$PROCESS(DEVICE$READY)=TRUE;      AUT16430
                              ELSE                                         AUT16440
                              WRITER$PROCESS(DEVICE$READY)=FALSE;          AUT16450
                    END;                                                   AUT16460
               ELSE                                                        AUT16470
                    DO;                                                    AUT16480
                         IF NOT INPUT(4) THEN                              AUT16490
                              WRITER$PROCESS(DEVICE$READY)=TRUE;           AUT16500
                              ELSE                                         AUT16510
                              WRITER$PROCESS(DEVICE$READY)=FALSE;          AUT16520
                    END;                                                   AUT16530
          END;                                                             AUT16540
                                                                           AUT16550
     IF NOT ROR(INPUT(CRT$STATUS),2) THEN                                  AUT16560
          OPERATOR$OUTPUT$PROCESS(DEVICE$READY)=TRUE;                      AUT16570
     ELSE OPERATOR$OUTPUT$PROCESS(DEVICE$READY)=FALSE;                     AUT16580
                                                                           AUT16590
IF VIRTUAL THEN                                                            AUT16600
     DO;                                                                   AUT16610
     WRITER$PROCESS(DEVICE$READY)=TRUE;                                    AUT16620
     TRANSMIT$LOGIC$PROCESS(DEVICE$READY)=TRUE;                            AUT16630
     OPERATOR$INPUT$PROCESS(DEVICE$READY)=TRUE;                            AUT16640
     OPERATOR$OUTPUT$PROCESS(DEVICE$READY)=TRUE;                           AUT16650
     END;                                                                  AUT16660
                                                                           AUT16670
END PCLI$DEVICES;                                                          AUT16680
                                                                           AUT16690
POLL$USAT: PROCEDURE;                                                      AUT16700
                                                                           AUT16710
     /* LEVEL 2 PROCEDURE */                                               AUT16720
                                                                           AUT16730
     /*  POLLS THE TRANSMIT SIDE OF THE USART TO SEE IF THE                AUT16740
          USART IS READY TO SEND THE NEXT BYTE  */                         AUT16750
                                                                           AUT16760
                                                                           AUT16770
     IF ROR(INPUT(USART$STATUS),2) THEN /* USAT NOT READY */               AUT16780
          TRANSMITTER$PROCESS(DEVICE$READY)=FALSE;                         AUT16790
     ELSE  /* USAT IS READY */                                             AUT16800
```

```
        TRANSMITTER$PROCESS(DEVICE$READY)=TRUE;                          AUT16810
                                                                         AUT16820
                                                                         AUT16830
IF VIRTUAL THEN                                                          AUT16840
    TRANSMITTER$PROCESS(DEVICE$READY)=TRUE;                              AUT16850
                                                                         AUT16860
END PCLI$OSAT;                                                           AUT16870
                                                                         AUT16880
                                                                         AUT16890
    /*   ***********************************************    */          AUT16900
    /*   **********    PROGRAM LEVEL 1    **********    */              AUT16910
                                                                         AUT16920
                                                                         AUT16930
    /*  EXECUTIVE SCHEDULER AND DEVICE MANAGER (NON-INTERRUPT            AUT16940
        VERSION).  FIRST EXECUTABLE STATEMENT IN THE PROGRAM.  */        AUT16950
                                                                         AUT16960
                                                                         AUT16970
VIRTUAL=TRUE;                                                            AUT16980
RESTART:  CALL INITIALIZE;  /* INITIALIZE ALL PROGRAM VARIABLES         AUT16990
                               AND RETURN HERE FOR RESTART */            AUT17000
                                                                         AUT17010
DO FOREVER;                                                              AUT17020
    /* THE PROGRAM WILL LOOP AS LONG AS THE CPU IS FUNCTIONING          AUT17030
       AND ATTEMPT TO PROCESS AUTODIN MESSAGES                  */       AUT17040
                                                                         AUT17050
    IF NOT VIRTUAL THEN /* SLOW DOWN FOR THE TTY */                      AUT17060
        CALL TIME(250);                                                  AUT17070
                                                                         AUT17080
    CALL POLL$DEVICES; /* CHECK THE PERIPHERAL DEVICES */                AUT17090
                                                                         AUT17100
    CALL POLL$USAR; /* CHECK THE RECEIVE SIDE OF THE USART */            AUT17110
                                                                         AUT17120
    /* IF SYNCH HAS NOT BEEN ACHIEVED, THEN THE RECEIVE                 AUT17130
       LOGIC PROCESS IS SCHEDULED TO ACHIEVE SYNCH.                      AUT17140
       IF IT'S DEVICE IS READY THEN IT MUST PROCESS AN                   AUT17150
       INCOMING BYTE NO MATTER WHAT IT'S STATUS IS  */                   AUT17160
                                                                         AUT17170
    IF RECEIVE$LOGIC$PROCESS (SCHEDULED) OR                              AUT17180
       RECEIVE$LOGIC$PROCESS (DEVICE$READY) THEN                         AUT17190
            CALL RECEIVE$LOGIC;                                          AUT17200
                                                                         AUT17210
    /* IF INCOMING MESSAGE TRAFFIC NEEDS TO BE WRITTEN ON               AUT17220
       THE SELECTED OUTPUT DEVICE, THEN THE WRITER PROCESS               AUT17230
       WILL BE SCHEDULED                                        */       AUT17240
                                                                         AUT17250
    IF WRITER$PROCESS(SCHEDULED) THEN                                    AUT17260
            CALL WRITER;                                                 AUT17270
                                                                         AUT17280
```

```
/* IF THE OPERATOR INPUT PROCESS DEVICE IS READY                    AUT17290
   IT MEANS THE HUMAN OPERATOR DESIRES TO INPUT                     AUT17300
   SOME INFORMATION TO THE PROGRAM                 */               AUT17310
                                                                    AUT17320
IF CPERATOR$INPUT$PROCESS (DEVICE$READY) THEN                       AUT17330
     CALL OPERATOR$INPUT;                                           AUT17340
                                                                    AUT17350
/* IF AN ALARM NEEDS TO BE SENT TO THE HUMAN OPERATOR               AUT17360
    THEN THE OPERATOR OUTPUT PROCESS WILL BE SCHEDUL-               AUT17370
    ED.  IF THE DEVICE IS ALSO READY, THEN THE PROCESS              AUT17380
    IS PERMITTED TO RUN.                                 */         AUT17390
                                                                    AUT17400
IF OPERATCR$OUTPUT$PROCESS (SCHEDULED) AND                          AUT17410
     OPERATOR$OUTPUT$PROCESS (DEVICE$READY) THEN                    AUT17420
          CALL OPERATOR$OUTPUT;                                     AUT17430
                                                                    AUT17440
IF NOT VIRTUAL THEN /* SLOW DOWN FOR THE TTY */                     AUT17450
     CALL TIME(250);                                                AUT17460
                                                                    AUT17470
CALL POLL$USAT; /* CHECK THE TRANSMIT SIDE OF THE USART */         AUT17480
                                                                    AUT17490
/* IF TRANSMIT LOGIC'S DEVICE IS READY AND WE ARE                   AUT17500
   SENDING A MESSAGE, THEN WE MUST PROCESS AN                       AUT17510
   CUTGOING BYTE.                                    */             AUT17520
                                                                    AUT17530
IF TRANSMIT$LOGIC$PROCESS(DEVICE$READY) AND SENDING THEN            AUT17540
     CALL TRANSMIT$LOGIC;                                           AUT17550
                                                                    AUT17560
/* IF TRANSMIT PROCESS'S DEVICE IS READY THEN WE                    AUT17570
   MUST SEND SOMETHING TO THE TRANSMIT SIDE OF THE                  AUT17580
   USART, EVEN IF IT IS ONLY SYN PATTERN             */             AUT17590
                                                                    AUT17600
IF TRANSMITTER$PROCESS (DEVICE$READY) THEN                          AUT17610
          CALL TRANSMITTER;                                         AUT17620
                                                                    AUT17630
                                                                    AUT17640
IF BYTE$COUNTER=60000 THEN                                          AUT17650
     DO;                                                            AUT17660
          BYTE$COUNTER=0;                                           AUT17670
          TEST$COUNTER=TEST$COUNTER + 1;                            AUT17680
          IF TEST$COUNTER=3 THEN HALT;                              AUT17690
     END;                                                           AUT17700
                                                                    AUT17710
 USAR$CHECK=USAR$CHECK + 1;                                         AUT17720
 IF USAR$CHECK > 250 THEN CALL ALARM(11);                           AUT17730
 /* WE'VE LOOPED 250 TIMES WITHOUT GETTING A RECEIVED BYTE -       AUT17740
 SCMETHING IS OBVIOUSLY WRONG - - CALL ALARM.            */         AUT17750
                                                                    AUT17760
```

```
END;      /* OF THE DO FOREVER LOOP  */                          AUT17770
                                                                 AUT17780
EOF                                                              AUT17790
```

BIBLIOGRAPHY

1. Defense Communications Agency Circular 310-D70-30, DCS AUTODIN Switching Center and Tributary Operations, June 1970.

2. Defense Communications Agency Circular 370-D175-1, DCS AUTODIN Interface and Control Criteria, October 1970.

3. Hartmanis, J. and Stearns, R. E., Algebraic Structure Theory of Sequential Machines, Prentice-Hall, 1966.

4. Intel Corporation, Intel 8080 Microcomputer Systems User's Manual, 1976.

5. Intel Corporation, Intellec 8/Mod 80 Microcomputer Development System Reference Manual, 1974.

6. Intel Corporation, Intellec 8/Mod 80 Operators Manual, 1974.

7. Intel Corporation, 8008 and 8080 PL/M Programming Manual, revision A, 1975.

8. Intel Corporation, 8080 PL/M Compiler Operators Manual, revision A, 1975.

9. Intel Corporation, 8080 Simulator Software Package, 1974.

10. Joint Chiefs of Staff Publication JANAP 128(E), Automatic Digital Network (AUTODIN) Operating Procedures, June 1973.

11. Madnick, S. E. and Donovan, J. J., Operating Systems, McGraw-Hill, 1974.

12. Renninger, J. F., Reduction of AUTODIN/NIDN Line Disciplines to Programmed Control Logic, Naval Postgraduate School Master's Thesis, June 1975.

INITIAL DISTRIBUTION LIST

		No. Copies
1.	Defense Documentation Center Cameron Station Alexandria, Virginia 22314	2
2.	Library, Code 0212 Naval Postgraduate School Monterey, California 93940	2
3.	Chairman, Code 52 Computer Science Department Naval Postgraduate School Monterey, California 93940	1
4.	Professor V. Michael Powers, Code 52Pw Naval Postgraduate School Monterey, California 93940	2
5.	Professor Uno R. Kodres, Code 52Kr Naval Postgraduate School Monterey, California 93940	1
6.	Captain Gordon E. Anderson, USMC 2125 Mountain Vista Drive Encinitas, California 92024	1
7.	Mr. Kenneth R. Ausich Building 110 Stanford Research Institute 333 Ravenswood Avenue Menlo Park, California 94025	1
8.	Commandant of the Marine Corps Headquarters, U.S. Marine Corps (Code CCPP-11) (Attn: Major Beeler) Washington, D. C. 20380	1